2008-06

Implementing an intrusion dectection [sic] detection system in the MYSEA architecture

Tenhunen, Thomas F.

Monterey, California. Naval Postgraduate School

NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

IMPLEMENTING AN INTRUSION DETECTION SYSTEM IN THE MYSEA ARCHITECTURE

by

Thomas Tenhunen

June 2008

Thesis Advisor:	Cynthia E. Irvine
Co-Advisor:	Thuy D. Nguyen

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2008	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE Implementing an Intrusion Detection System in the Mysea Architecture	**5. FUNDING NUMBERS**
6. AUTHOR(S) Thomas Tenhunen	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The Monterey Security Architecture (MYSEA) supports a multilevel secure (MLS) network and a number of single level networks at different classification levels. The MYSEA MLS server is the focus of policy enforcement. It implements a Dynamic Security Services mechanism (DSS) that can modulate IPsec security attributes and MYSEA security services based upon administrator choices. Use of intrusion detection technology on the unprotected single level networks can provide administrators with actionable information to inform DSS choices.

The objective of this thesis is to design an intrusion detection system (IDS) architecture that permits administrators operating on MYSEA client machines to conveniently view and analyze IDS alerts from the single level networks.

A progressive set of analyses and experiments was conducted that led to a working implementation of an IDS for MYSEA. Sensors are located on the single level networks. Their alerts are fed into the MLS server, where single level databases are used to store and organize the data. Administrators can login from the MLS LAN and examine IDS results, which may be used to derive new DSS policies. A testing methodology was developed and functional tests were performed. Implementation considerations for future extensions of this work are presented.

14. SUBJECT TERMS Type Keywords Here Intrusion Detection Systems (IDS), Information Assurance(IA), Monterey Security Architecture (MYSEA), Global Information Grid (GIG)			**15. NUMBER OF PAGES** 169
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

THIS PAGE INTENTIONALLY LEFT BLANK

IMPLEMENTING AN INTRUSION DECTECTION SYSTEM IN THE MYSEA ARCHITECTURE

Thomas F. Tenhunen
Civilian, Naval Postgraduate School
B.S., Chapman University, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2008

Author: Thomas F. Tenhunen

Approved by: Cynthia E. Irvine, Ph.D.
Thesis Advisor

Thuy D. Nguyen
Co-Advisor

Peter J. Denning, Ph.D.
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Monterey Security Architecture (MYSEA) supports a multilevel secure (MLS) network and a number of single level networks at different classification levels. The MYSEA MLS server is the focus of policy enforcement. It implements a Dynamic Security Services mechanism (DSS) that can modulate IPSec security attributes and MYSEA security services based upon administrator choices. Use of intrusion detection technology on the unprotected single level networks can provide administrators with actionable information to inform DSS choices.

The objective of this thesis is to design an intrusion detection system (IDS) architecture that permits administrators operating on MYSEA client machines to conveniently view and analyze IDS alerts from the single level networks.

A progressive set of analyses and experiments was conducted that led to a working implementation of an IDS for MYSEA. Sensors are located on the single level networks. Their alerts are fed into the MLS server, where single level databases are used to store and organize the data. Administrators can login from the MLS LAN and examine IDS results, which may be used to derive new DSS policies. A testing methodology was developed and functional tests were performed. Implementation considerations for future extensions of this work are presented.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to express my sincere thanks to my advisors, Dr. Cynthia Irvine and Thuy Nguyen, for their time, effort, and patience with their help over the course of this project. Their expertise and guidance was invaluable and without it, this project would never have been completed. I would also like to thank Jean Khosalim for his technical assistance, time and patience going through the installation instructions and test procedures.

This material is based upon work supported by the National Science Foundation, under grant No. CNS-0414102. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Intrusion Detection Systems (IDS) are an important component of a comprehensive defense in depth approach for securing information system networks by looking for malicious activity on the network. The function of the IDS is to watch and assess network activity without affecting that traffic. Data collected by the IDS is sent to a database for further analysis by skilled security analysts and system administrators. These analysts often use tools that can parse the captured data into meaningful human readable formats. IDS can really be viewed as a collection of three main software components, a sensor that resides on a network segment assessing traffic, a database for storing the results of traffic assessments in the form of alert data sent by the IDS, and an analysis tool for examining the collected data.

Deployment of IDS sensors in a network at a single classification level is straightforward. To monitor inbound and outbound network traffic sensors are placed at the demarcation points of the enclave. Data sent from the IDS sensors to a central repository for analysis provide analysts the capability to determine the criticality of this data and take appropriate actions. Department of Defense (DoD) networks have different classification levels, for example the NIPRNet (Unclassified) and SIPRNet (Secret) are military networks that are separated by a guard or physically separated. Data generated by an IDS on the NIPRNet cannot be stored with data generated by an IDS on the SIPRNet since in both cases the IDS data came from a sensor that carries the same classification level as the network it monitored. This separation of data presents a problem when analysts need to perform correlation analysis on alerts. With data separated by classification level an analyst is really only working with a sub-set of data. Combining the data would provide the analyst will a full set of data to work with, which would give comprehensive results from analysis performed on the data.

The Monterey Security Architecture (MYSEA) is an experimental multilevel secure computing environment that can provide a common end-point between networks of different classification levels [1]. In the MYSEA environment, a multilevel secure operating system installed on an Intel hardware platform provides isolation by enforcing Mandatory Access Control (MAC) policies. This platform is the MYSEA server. A MYSEA server also acts as a redistribution point for clients inside the MYSEA network that need to access resources such as web and email.

Collecting and storing data sent by the IDS from the multiple networks at different classification levels presents a unique challenge. Some of the data contains the payload of the packet that triggered the alert. By default, the classification of the alert is the same as the classification of the network from which the alert was generated. Despite this challenge, there is also a unique opportunity to have all of the alert notifications from networks at different classification levels fused as actionable information. A unique component of the MYSEA environment is its Dynamic Security Services (DSS), which can provide a Quality of Security Service (QoSS) [2]. In particular, the current implementation allows IPSec security associations [2] to be modulated based upon session attributes and network conditions. The motivation behind a QoSS was to raise the security level based on perceived network threats, but there must be a way to identify these threats. Using an IDS and storing alerts for analysis will provide a means to identify attacks that can threaten the network and if the attacks warrant a change in IPSec security requirements the analyst can inform the administrator for the DSS.

B. PURPOSE OF STUDY

The objective of this thesis is to study the best method for deploying an Intrusion Detection System and setting up a central repository for alerts in a MLS architecture. A secondary objective is to provide a method for the DSS administrator to view attacks to determine if switching to a different Dynamic Security Services (DSS) security policy is warranted. This thesis examines the following questions.

1. In the MYSEA environment, what is the best way to locate and configure IDS sensors and a central repository to collect and store sensor data to afford the system analysts with a complete picture?

2. If data cannot be stored in one central database due to unforeseen complications what are reasonable alternatives?

The IDS field is very active with many feasible products ranging from open source products to high-end innovative commercial products. To provide an organizational framework, the following methodology is used in this research effort. Background material will be established by conducting a review of existing products and researching previous work in the field of intrusion detection. General requirements coupled with selection criteria will be applied to a formal selection process to cull IDS components appropriate for the prospective architecture. A new architecture will be developed based on the current MYSEA architecture for integrating the new IDS components that addresses security issues concerned with moving data at different classifications in a MLS environment. Proving the concepts will first take place by conducting experiments on Linux systems that are similar to the XTS400 system [3]. These experiments, if successful, will lead to the development of a functional prototype that can be employed in the MYSEA architecture. To verify that the integration of the IDS components functions correctly in the MYSEA environment, testing methodologies will be developed to exercise individual components as well as to exercise the entire suite as one functional unit. The results from this research project will then be used to analyze the utility of the IDS components for presenting a picture of what is happening on the network with respect to possible attacks and how this data can be used to inform decision makers. .

C. ORGANIZATION OF THESIS

Organization of this thesis is as follows:

Chapter I provides the motivation and purpose of this thesis, and introduces the problem of implementing an IDS in a multilevel security environment

Chapter II provides detailed background information on different components of an Intrusion Detection Systems with a more in depth review of Snort. Other topics covered include the MYSEA network, analysis tools used to analyze captured data such as the Basic Analysis Security Engine (BASE).

Chapter III covers the selection process for choosing an IDS, database and analysis tool that will be implemented in the MYSEA architecture.

Chapter IV covers the concept of operations and design of the entire IDS system from sensors to its analysis tool.

Chapter V describes the implementation of the actual components on both Red Hat 8 Linux system and a XTS 400 system running the STOP 6.3 OS. The chapter concludes with the test plan and testing results for both environments

Chapter VI consummates the thesis with conclusions and future work.

II. BACKGROUND

This chapter contains background information on the Monterey Security Architecture (MYSEA), Intrusion Detection Systems (IDS), analysis tools used in examining IDS data, and a proxy service for the open source database MySQL.

A. MYSEA ENVIRONMENT OVERVIEW

The Department of Defense (DoD) maintains physical separation between networks of different classification levels. For example, there are the unclassified NIPRNet and the secret SIPRNet. For both networks to exist in the same enclave, each network must be built as a physically separated infrastructure, therefore doubling equipment such as workstations, servers, routers, etc. The Monterey Security Architecture (MYSEA) supports a common infrastructure for security classification levels on a single network by using high assurance components that enforce mandatory security policies [1]. This multilevel secure network environment consists of both high-assurance components such as the XTS-400 server running a secure operating system and low-assurance commercial components. A trusted communications path is setup between these components using the Trusted Path Extension (TPE) module. The TPE provides a trusted path between users who are computing on common COTS operating systems and MYSEA high assurance server. It should be noted that the TPE is part of the distributed Trusted Computer Base (TCB) along with the MYSEA server. All internal network traffic in the MYSEA environment uses IPSec[1]. Dynamic Security Services (DSS) provide a method for adjusting Security Associations (SAs) for confidentiality and integrity managed by IPSec [2]. This architecture also supports multiple external networks that are at different classification levels.

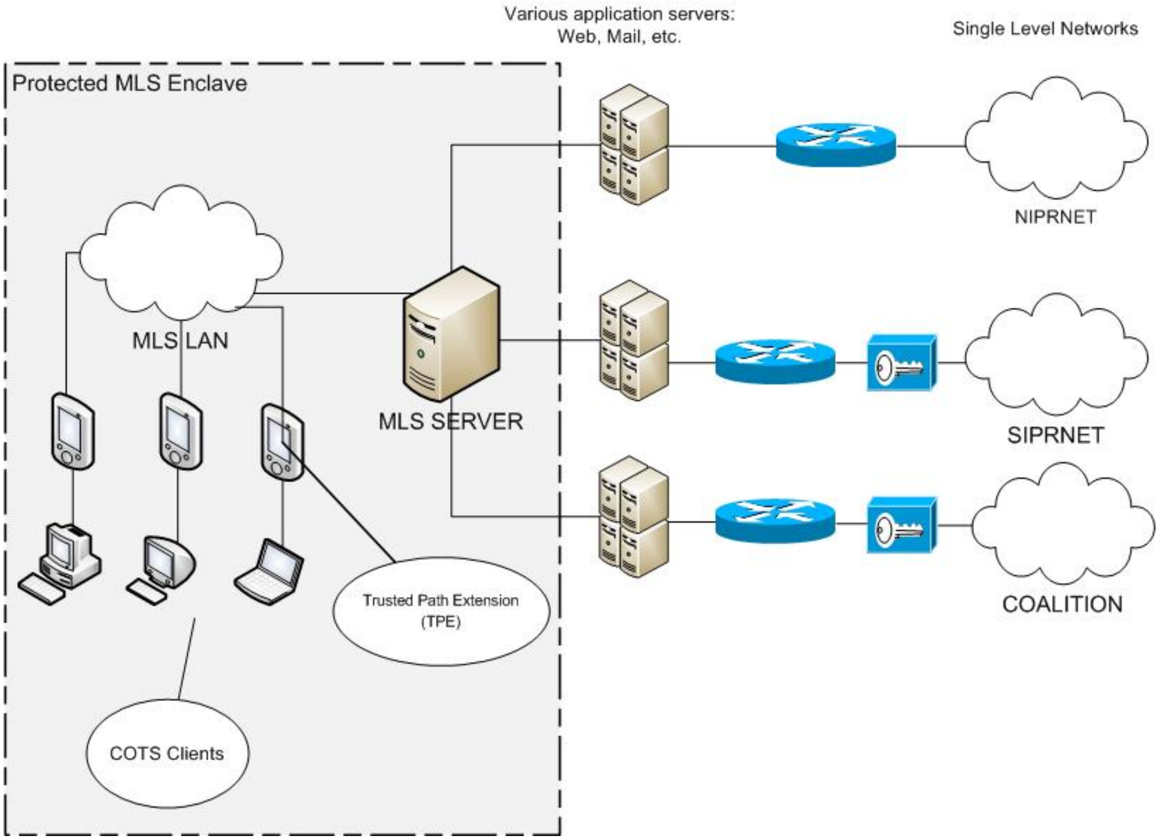


Figure 1: MYSEA Environment

1. XTS-400 and STOP Operating System

The keystone of the MYSEA environment is the XTS-400 platform running the Secure Trusted Operating System (STOP) developed by BAE Systems. This operating system can enforce Mandatory Access Control policies (MAC) to enforce confidentiality and integrity. The MAC policies are based on the Bell and LaPadula model [3] for confidentiality and Biba model [3] for integrity. Enforcement of these policies on a single system allows for simultaneous processing and storage of data at different classification levels by users with different clearances [3].

To protect the security kernel, the STOP uses a method known as “domains of isolation” [3] with the security kernel being at domain 0, the most protected domain. Note that in most descriptions of such systems, these are known as privilege domains. Processes are restricted by domains and can only communicate with domains of equal or lesser privileges. There are four domain levels, 0 through 3. Level 0 is for the Security Kernel, level 1 for Trusted System Services, level 2 for Operating System Services and level 3 for Application Programs. Figure 2 shows a diagram of how the domains interact with each other.

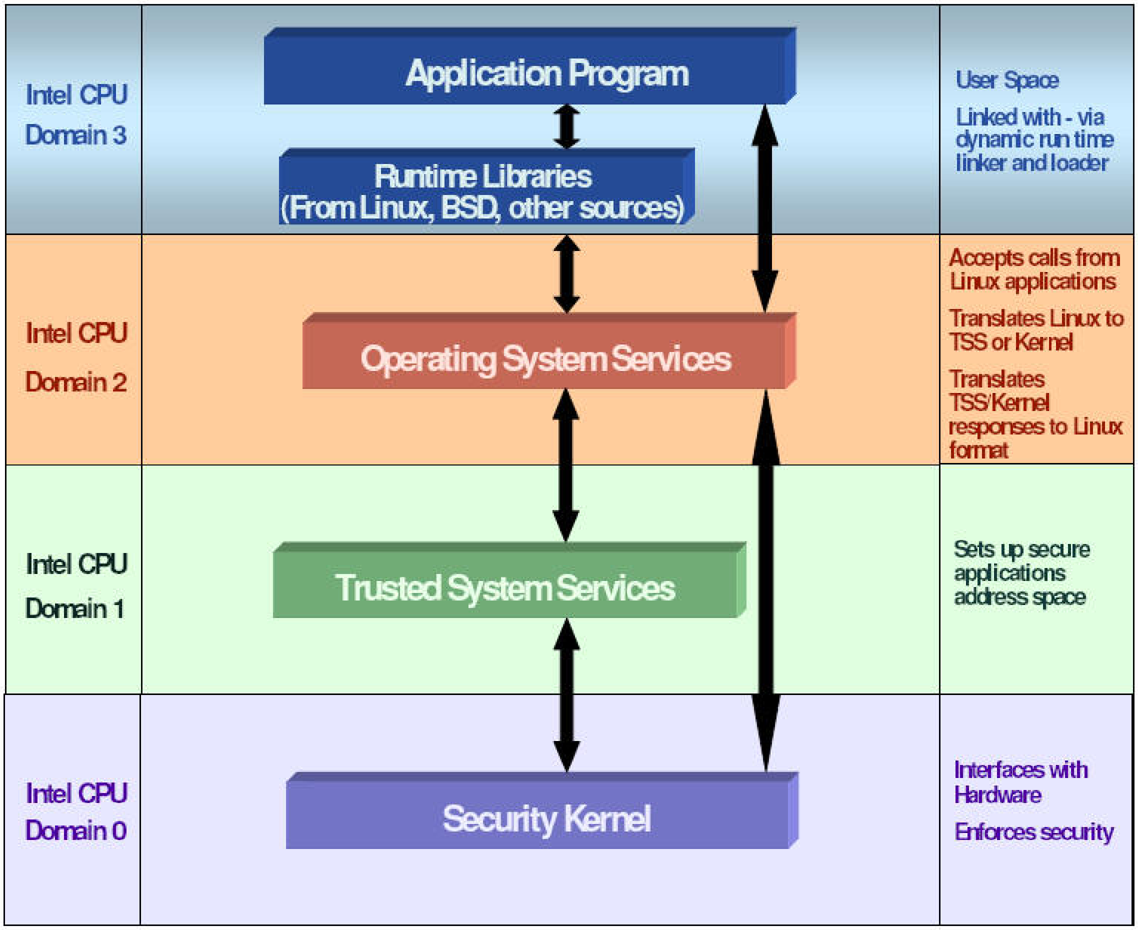

Intel CPU Domain 3
Application Program
Runtime Libraries (From Linux, BSD, other sources)
User Space
Linked with - via dynamic run time linker and loader
Intel CPU Domain 2
Operating System Services
Accepts calls from Linux applications
Translates Linux to TSS or Kernel
Translates TSS/Kernel responses to Linux format
Intel CPU Domain 1
Trusted System Services
Sets up secure applications address space
Intel CPU Domain 0
Security Kernel
Interfaces with Hardware
Enforces security

Figure 2: STOP Four-Domain Architecture [3]

2. Dynamic Security Services (DSS)

The Dynamic Security Services provide a method to balance protection for communications in the MYSEA environment to meet external conditions. A dynamic service allows the IPSec policy to be changed based on the state of the operating environment while preserving finite resources for the computer system [2]. Currently to change the System Security Level and System Operation Mode, IPSec policies on the DSS requires manual intervention. The adjustment of these IPSec policies stems from John Horn's work with Dynamic Security Services [2]. Communication channels may service connection requests at different classification levels using an appropriate cryptographic algorithm such as MD5, SHA1, DES, and so on, in order to balance CPU load with the required level of classification [2]. For example, if System Security Level is set to Low and System Operational Mode is set to Normal then the AH policy would be set to use the MD5 algorithm and the ESP policy set to use the DES algorithm. Escalating either System Security Level or System Operational Mode would increase the algorithm used by AH or ESP, or both. Ideally, it is more efficient to notify the MYSEA server of changing conditions using an automated process. Prior work by John Horn leaves this problem as future work [2]. There is also no method in the MYSEA framework to monitor for the occurrence of network attacks. The operator or the DSS needs to have an indicator to forecast network conditions. An IDS can be tied in with the DSS to serve as that indicator.

3. External Networks

Depending on the physical location of a MLS environment, it may have to communicate with single level networks at various classification levels. Figure 1 shows the MYSEA MLS enclave networked with the unclassified NIPRNET, secret SIPRNET, and another arbitrary network, COALITION. One of the functions of the NIPRNET is to handle normal Internet traffic for users in the MLS LAN. Therefore, this network also poses the greatest risk with respect to malicious traffic.

Interconnection of two networks at two differing classifications levels is handled by the MYSEA server by enforcing MAC policies. Even though two networks of different classifications levels terminate at the same physical server, the MLS server is capable of keeping network stacks at different classification levels logically separated with high assurance, thus eliminating the use of a guard or requiring physical separation.

4. Trusted Path Extension

The Trusted Path Extension (TPE) is a logical extension of the MYSEA server. It is a small device attached to the client that provides a secure, non-forgeable connection between the Trusted Computing Base (TCB) of the MYSEA server and the user (see Figure 1). The TPE provides for user authentication and establishing users' session classification level [1].

5. MLS HTTP Proxy Service

The MYSEA server has a HTTP Proxy service that handles clients requesting web pages of varying classification levels. A proxy server services the requests of its clients by forwarding requests to other servers. On the MYSEA MLS server, the HTTP proxy service handles all web requests from its clients. To save time in writing a trusted application this project will take advantage of this HTTP proxy service by using a web-based intrusion analysis tool. Thus, analysts can log in on any workstation connected to a TPE and gain access to the IDS analysis web site.

B. INTRUSION DETECTION SYSTEMS (IDS)

This section describes how an IDS functions and why they are used to defend networks. There are several open source and commercial off the shelf IDS products available. Chapter III will cover the criteria for choosing an IDS for this project.

1. Overview of an IDS

An Intrusion Detection System acts like a sentinel with the ability to look for activity that resembles reconnaissance, attempted system compromise, or other malicious activity [4]. An IDS's job is to inspect the contents of network traffic for possible attacks. IDSes are capable of parsing through network traffic and comparing that traffic to known attack signatures. When IDSes discover an attack, they can raise an alarm by sending out alerts.

In 1980, James P. Anderson described the notion of IDS in his paper Computer Security Threat Monitoring and Surveillance [6]. The idea was to build a better security audit tool designed more for the security administrator. Dr. Dorothy Denning's paper published in 1987, An Intrusion-Detection Model, served as the first blueprint for designing an IDS [7]. Her model addresses a wide range of threats from outside attackers to abuses by insiders.

IDS architectures consist of three different components, the sensors, repositories, and analysis tools. A preferred architecture has each component installed on separate hardware platforms connected by a high speed LAN, but installation of all components on one platform is acceptable provided the hardware resources can handle the load. Figure 3 shows a typical IDS architecture.

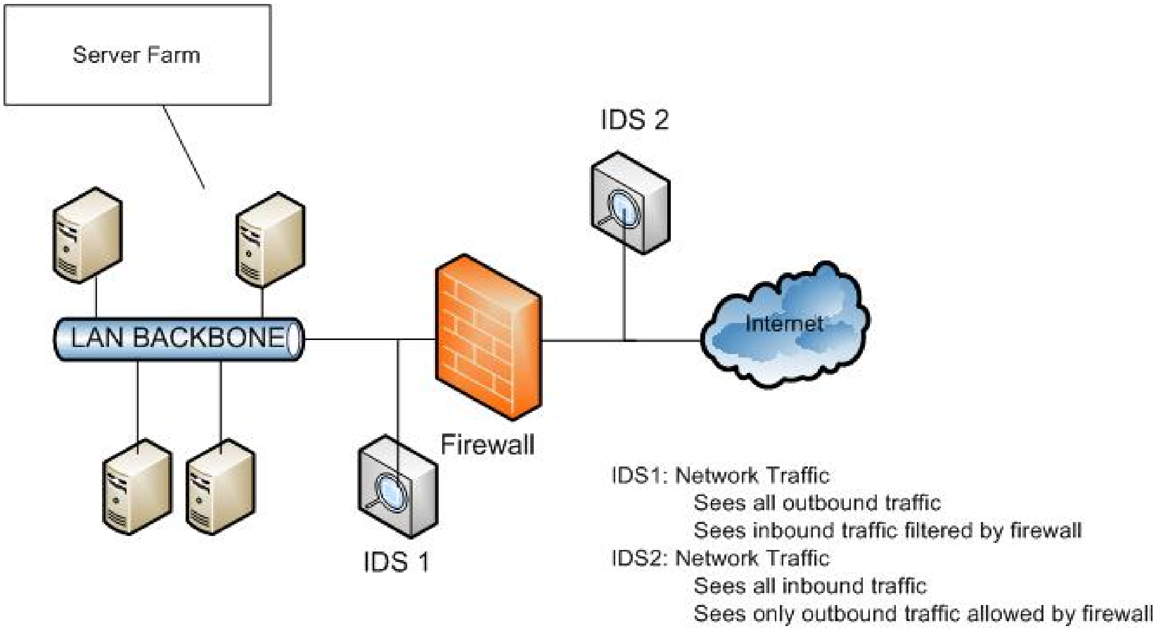


Figure 3: Generic Network IDS Placement

Effective monitoring of network traffic relies on sensor location. Sensors are placed at strategic network locations that will allow it to see as much network traffic as possible on that network segment. Sensors could be located in front of firewalls, behind them, or on a special zone known as a Demilitarized Zone (DMZ). A sensor placed in front of a firewall will see all of the network traffic on that segment; however, the amount of network traffic could overwhelm the sensor. Sensors behind a firewall or on a DMZ monitor filtered traffic that the firewall allowed. Sensors 'sniff' network traffic by capturing all network traffic through the use of a Network Interface Card (NIC) set in promiscuous mode.

Normally a NIC operates in non-promiscuous mode and only forwards traffic up the stack if it is destined for the platform's media access control (MAC) address or it is a broadcast. This method cuts down on the amount of traffic processed by the operating system. When operating in promiscuous mode, the host will forward all traffic up the TCP stack. IDS sensors utilize this feature to analyze all traffic that can be seen on the network segment to which they are attached. A promiscuous NIC does not even require an IP address and usually does not have one. This technique provides additional security for the sensor. Communications between sensors and repositories takes place over a separate maintenance network.

Analysis of network packets consists of one of two methods. The first method identifies known attacks by looking for a distinct signature. Signatures are unique lines of machine code or strings found in the payload of network packets. For example, a common method to inject shell code uses repeated occurrences of 0x90 in the payload, which is machine code for a NOP (short for perform no operation). The use of numerous 0x90 in this example indicates a signature known as a "nop slide". The other method of attack detection is anomaly detection, which relies on complex statistics to build a set of rules. These rules are used similar to signature rules since they form the basis for generating alerts when possible attacks are detected. Analysts first create a baseline of the network to determine acceptable normal behavior. Any network activity that falls outside of this acceptable behavior is flagged as an alert.

Neither of these methods is perfect, and they will create false positives and false negatives. With respect to IDS, a false positive occurs when the IDS signals that there is an attack when, in fact, no attack has taken place. This can result from the use of poorly designed rules or from bad statistical measurements of the network. There is another category of false positives that result when the IDS detects a true attack but none of the systems in the network are vulnerable to this attack. False positives raise unnecessary alarms that detract from the purpose of using IDS. False positives can be managed by using well-written rules and removing obsolete rules. On the other hand, false negatives occur when the IDS misses events. Reasons for missing events can vary, from hackers using evasion techniques, network packet loss on a busy network, or using improper rules. Since false negatives equate to not seeing attacks, this creates a situation that is not desirable. The only real counter measure to managing false negatives is to follow all published best practices when deploying sensors.

A default configuration for IDS sensors results in the storage of alerts in a file using some kind of unified log format. Log files could potentially hold thousands, even millions of pieces of information. An analyst would be overwhelmed trying to make sense of such log files. To overcome this problem, sensor configurations allow for output software plug-ins that can send alerts and logs to a repository. A central repository provides a place to keep alerts and logs collected from all sensors. Collecting all the alerts at one location gives the security analyst a complete picture of what is happening on the network. These repositories are common relational database products such as MySQL, Microsoft MSSQL and PostGreSQL. Using a relational database is more advantageous than using log files since database products come with a wide assortment of management tools such as automated backups and replication to other databases.

Analysis tools provide a picture of what is happening on the network. Some IDS products come with their own proprietary tools, whereas open source IDS have numerous add-on tools to choose from. The basic function of these tools is to pull information from relational databases and compile it into meaningful statistics, correlation information, charts, and graphs, therefore making it easier for security analysts to make decisions about possible attacks.

2. Snort

Presently there are several open source and commercial-off-the-shelf Intrusion Detection System applications. Over the last nine years, the market has expanded and many of the industry leaders such as Cisco, McAfee, Symantec, Juniper Networks, etc. have rolled out either an IDS or the next generation Intrusion Prevention System (IPS). An IPS has the same capabilities as IDS, but with the added ability to block the incoming attacks that it detects. [9]. An IPS is normally placed in line with the flow of traffic, which means all traffic physically flows through the IPS. The IDS can only passively monitor traffic flows, traffic does not flow into an IDS and back out to its destination. Snort's first release was December 22, 1998 by Marty Roesch, as a packet sniffer tool [4]. Snort began using signature-based analysis in January 1999 [4]. Today Snort is the leading open source IDS and enjoys large community support. Most of the support comes from Roesch's security company Sourcefire.

Snort's architecture is composed of four basic components:

- A network packet sniffer
- The preprocessor
- The detection engine
- Output plug-ins (alert file, log file, database)

A packet sniffer captures all of the network traffic off the wire. Snort's packet sniffer uses the libpcap library, a freely distributed library used for grabbing packets off a network. Most Linux systems come with libpcap installed. The Windows operating systems use winpcap with Snort.

Packets grabbed off the wire are just raw packets, so the job of the preprocessor is to perform sorting and checking to facilitate efficient analysis. Snort's engine uses compiled code called *preprocessors* to normalize traffic and examine the traffic for possible attacks or abnormal behavior. The first plug-in for example is a protocol decoder. Decoding of all network traffic must take place before passing it on to the other preprocessors and the detection engine. Other plug-ins handle IP defragmentation, stateful inspection and application layer protocols. The ability of preprocessors to find attacks or abnormal behavior saves further processing time and valuable system resources. Creating new preprocessors or extending existing preprocessors requires C programming skill but documentation is available if either task is required.

After data leaves the preprocessor, it is handed off to the detection engine. Snort is primarily a rules-based IDS. The detection engine runs the data through a set of rules that look for distinct signatures. Data packets that match a rule will trigger the alert pre-processor. Signature rules are categorized into groups such as those that can identify Trojan horses, buffer overflows and other application vulnerabilities. Rules have their own syntax so administrators must be careful when writing additional rules. One syntax error in any rule will prevent the Snort process from starting. Currently there is no alert process in place to verify that Snort initialized properly. It is up to the administrator to check the systems' log files to verify Snort initialization. If a rule prevented Snort from starting, the log file will state which rule failed. Sourcefire normally releases new rules, which have been tested and certified to work correctly, however anyone can design new, customized rules.

By default, Snort writes its alerts to an alert file on the sensor. Alert and logging output are easily configured using Snort’s configuration file. Some of the most common output methods are to send alerts to a syslog process or to a relational database such as MySQL or Windows MSSQL server. There are quite a few other ways to output alerts and security administrators may program their own output methods such as sending alerts to non-supported databases or file formats.

C. IDS ANALYSIS TOOLS

This section will cover IDS analysis tools. These tools play an important role in helping analysts make sense of of the alert data gathered from various IDS sensors.

1. Overview of IDS Analysis Tools

An IDS sensor could collect thousands, even millions of alerts depending on the sensor's location on the network. An interview was conducted on November 14th, 2007, with two security personnel who work in the NPS Information Technology and Communication Services (ITACS) center. The objective was to ascertain how many alerts a production IDS sensor could accumulate. An interviewee explained that the IDS database held over two million alerts collected from just two sensors over a period of three months. Most of the alerts were not malicious in nature or had reasonable explanations, but the job of the analyst is to sift through these alerts looking for real attacks. This illustrates why it is imperative to use some type of analysis tool to parse through the voluminous Snort alert logs. Imagine trying to read an alert file with two million entries. For ITACS, the analysis tool chosen is an open source application called BASE, (Basic Analysis and Security Engine).

There are four important functions of IDS data analysis tools:

- Real-time alerting
- Attack Detection and Verification
- Incident Analysis
- Reporting

Real-time alerting requires setting up tools designed to monitor log files for specific patterns. In real-time alerting, the alert stream is examined for events based on certain combinations that express increasingly complex situations [4]. Implementing these tools is beyond the scope of this project. The remaining three factors take place offline on collected data.

Attack detection and verification is the process of separating false positives from useful data. An IDS is not perfect because it either relies on rules for known attacks or statistical data for anomalous attacks. Finding an attack does not mean that is was successful, but the analyst should start an incident analysis process. The process consists of determining when the attack occurred, what the attack did, what systems were compromised and how was the attack performed, i.e., what method was used, such as shell code injection, buffer overflow, etc. Reporting on the incident gives decision makers an idea of the impact on the systems that were compromised. Regular reporting also gives the analyst repeated measurements over time to help fine tune the IDS by tracking changes to the data.

2. Basic Security Analysis Engine (BASE)

BASE is one of the leading open source web-based applications used to analyze IDS data. Written in PHP, BASE works in conjunction with the Apache web server and the MySQL relational database [5]. It provides some excellent features to assist with the task of working with a large amount of Snort alerts. A graphical interface provides database searching and sorting of alerts by protocols, time of day, unique alerts, and other miscellaneous queries. BASE can display network stack Layer 3 (network) and Layer 4 (transport) packet information. Graphs and charts provide for the viewing of statistics based on specified parameters.

D. MYSQL PROXY SERVICE

This section covers a new program released from MySQL, an open source database company. To enhance functionality for database and application developers, MySQL designed a proxy service that works with MySQL clients and databases.

1. Overview

Proxy services are processes that service the requests of clients, by forwarding requests to other servers. By acting as an intermediary, all service requests appear to come from a single source. There are several types of proxies, but this project is using a new database proxy developed by MySQL.

2. MySQL Proxy Service

The MySQL proxy service is situated between one or more MySQL clients and a MySQL database server. Clients connect to the proxy using their regular credentials, instead of connecting directly to the server. This proxy acts the same as any other proxy process but with additional capabilities. Attached to the proxy service is a Lua interpreter. Lua is an interpretive scripting language, with an interpreter written in *clean* C, i.e., a common subset of ANSI C and C++. [10]. Using Lua scripts allows for flexibility in manipulating SQL queries before they reach the SQL server. Some of the new capabilities include injecting new data into a query, filtering, and even rewriting the query [8].

E. EMERALD

This section reviews EMEARALD, an intrusion detection and response project developed by SRI International [24].

1. Overview

Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD) is an advanced IDS project intended to provide IDS for distributed networks. The conceptual design is for a scalable surveillance and response architecture for large dispersed networks [24]. The design employs numerous monitors at various abstract layers in the network. The main advantage to EMERALD's architecture is the ability to detect not just local attacks, but also coordinated attacks such as distributed denial of service attacks or repeated patterns of attack against multiple domains [24]. Figure 4 shows the initial design of the EMERALD monitor architecture.

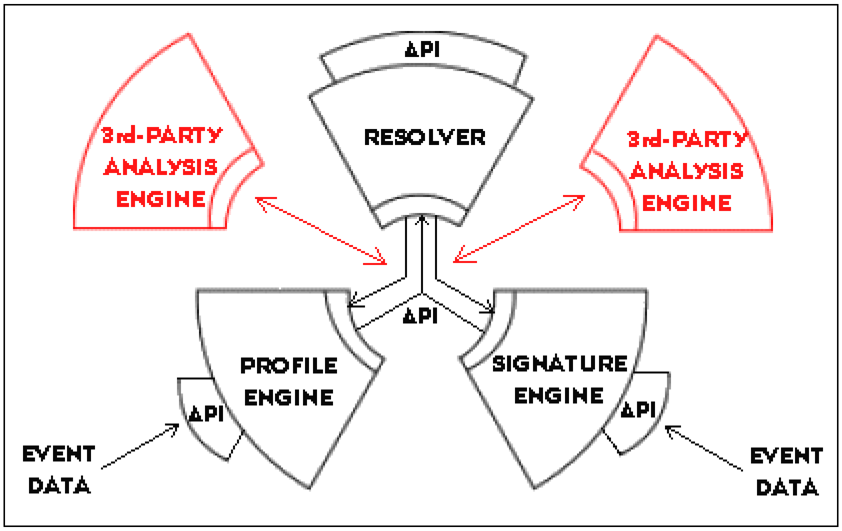


Figure 4: EMERALD Architecture [24]

A major strength of the EMERALD project is the design of components that cover the main areas of event detection and response such as signature detection, anomaly detection, and a correlation reporting method. EMERALD eBayes component performs anomaly detection through the use of probabilistic models of normal, attack, and anomalous behavior, and serves as the profile engine (see Figure 4) [25]. EMERALD eXpert is a highly targetable signature-analysis engine based on the expert system shell P-BEST [25]. This component serves as the signature engine (see Figure 4) to detect known malicious attacks.

The correlation component, Assessing Strategic Intrusions Using CIDF (ASIC), uses an analysis method developed by SRI and ISI/USC to correlate intrusion reports, discern large-scale patterns of attack, and infer the intent of the adversary [25]. The independent sensors use a standardized communication format, Common Intrusion Detection Framework (CIDF), as a means to communicate information amongst other sensors. If one sensor is under attack, it can communicate information about the attack to other sensors.

F. SUMMARY

Defense in depth for a unique environment like the MYSEA MLS network requires implementation of an IDS. The challenges associated with such an implementation lie with consolidating alerts from sensors that are at different classification levels and using an analysis tool to parse through this alert data. The next chapter describes the selection criteria used to choose an IDS for this project.

THIS PAGE INTENTIONALLY LEFT BLANK

III. CRITERIA FOR SELECTION OF IDS COMPONENTS

To formalize a selection method for choosing an IDS system, it is best to view an IDS in terms of its three major components: a sensor that reads network traffic, a database for storing alerts and logs, and analysis tools. This chapter will use the following approach to select the required components. The General Requirements section will list a few broad requirements related to the scope of the project in order to provide a starting point for generating specific criteria required for each component. To help pare down the available choices the selection criteria lists, in ranking order, criteria specific to this project. The selection process will explain, based upon considerations of criteria, how products were eliminated. The justification for selecting a product will be explained in the final section.

A. AVAILABLE IDS

There are numerous IDS available for purchase from software companies or freely downloadable as open source products.. Using the popular Internet search engines, such as Google, to find IDS products yielded numerous results touting new products. To choose an appropriate IDS sensor for this project, the following steps were performed.

1. General Requirements

Before reading numerous reviews of IDS or establishing specific criteria, general requirements based on the environment were established. These are listed here. First, the project will require multiple IDS sensors to handle the various single level networks that feed into the MYSEA architecture. To simplify the project only one type of IDS will be selected, however, it is possible to use multiple types of sensors. Second, the IDS sensors will not host a database to store events but instead will forward events to another database system. Third, the goal of this project is to establish a generic template for any future IDS products that would be used in the MYSEA architecture or any similar MLS environment. Fourth, the project has a limited budget and no time to deal with procurements. The procurement process would involve writing request for proposals (RFPs) and other such documents. Performing this level of work is beyond the scope of this project. Establishing these general requirements allows for the development of specific selection criteria. Each selection criteria, except for Open Source and Extensibility, will have a ranking system from 1 to 4, where 1 is the lowest rank and 4 is the highest. All of the criteria are equally weighted and the two criteria, Open Source and Extensibility are either "Yes" or "No", binary items. The ranking system gives quantifiable measures to subjective criteria.

2. Selection Criteria

- Open Source: The use of open source products saves time in dealing with proprietary software licenses, contracts and procurement processes. Open source products are freely distributed, operational products and can usually be downloaded in minutes. The term "open source" actually covers a wide range of license agreements; the most common is the GNU General Public License (GPL) [11]. Only open source IDS products that come with an open source license will be considered for selection. This ranking is a binary ranking of either a "Yes", or a "No".
- Platform compatibility: The IDS should be cross platform compatible with many Operating System platforms. As a minimum, the IDS should run on popular Linux and BSD distributions such as Debian, Fedora, and FreeBSD.

a. Ranking Platform Compatibility

1. Supports minimal complement of common operating systems

2. Supports some common operating systems

3. Supports many common operating systems

4. Supports most common operating systems

- Usability: Some software programs can be difficult to set up and configure for its intended use. This criterion looks at items such as how is the IDS setup and configured, if there are Graphical User Interfaces (GUIs) or if operation requires using Command Line Interfaces (CLIs).

b. Ranking Usability

1. Very difficult to setup and use, has CLI only

2. Moderately difficult to setup and use, has CLI only

3. Little difficulty in setup and use, has CLI only

4. Little to No difficulty in setup and use, has CLI and GUI

- Product maturity: One disadvantage to open source software is the body of work may not be fully functional or have too many bugs in it. Software of this nature is often labeled as ‘alpha’ or ‘beta’ code. These designations are a software engineering reference to the level of testing the program has undergone. The selection process will take into account software version numbers to help determine product maturity or the years in use. The reason for the ‘OR’ statement is that not all version numbers truly reflect the maturity of the product. For example, Wireshark, formerly known as Ethereal, is currently at version 1.0, but has been in use for about 10 years and is one of the most popular network protocol analyzers used today.

c. Ranking Product Maturity

1. Product is in alpha code or in use less than 1 year

2. Product is in beta code or in use less than 2 years

3. Low versioning numbers (< 1.0) or in use less than 3 years

4. Versions above 1.0 or in use greater than 3 years

- Support and documentation: Using good documentation aids in setting up a software program and using it correctly. Community forums and online documents are the main supply for getting support with open source products. Expert users of these programs set up web sites for others to share their experiences and knowledge. This selection criterion will factor in how much documentation is available, the quality of the documentation and if there are any tutorials that demonstrate how the product should work.

d. Ranking Support and Documentation

1. Very little documentation, no user forums or tutorials

2. Some documentation, has some user forums, no tutorials

3. Good documentation, has user forums, some tutorials

4. Good to excellent documentation, active user forums, several clear tutorials

- Extensibility: Having the source code allows the program to be modified or permits inclusion of new features to satisfy design requirements. It has not been determined at this point if modifications are required. Thus, this is a good selection criterion but is not highly weighted.

3. SELECTION PROCESS

Selecting an IDS can be reduced to only searching for open source products. A search of the Internet yielded some good sites that had information on IDS, but sites that were specific to the product were only used in research after reading reviews from non-biased sources such as www.linuxworld.com [12] and Infosecwriters.com [13]. From this research there are two leading open source IDS: Snort and Bro. Other open source products are used as Host-based Intrusion Detection Systems, which are not applicable to this project. Each product was set up in a small test environment to determine if they operated as an IDS. Elementary tests such as port scans and pre-generated network captures containing known exploits were used to determine usability. Each product's web site was reviewed to determine the other selection criteria [14], [15]. Table 1 summarizes the scoring for each system.

Selection Criteria

	Open source	Platform Compatibility	Usability	Product Maturity	Support & Documentation	Extensibility
Snort	✓	4	3	4	4	✓
Bro	✓	3	2	4	3	✓

Table 1: Selection Criteria for IDS

4. Selection Outcome

Snort and Bro perform the same functions but they detect intrusions using different methods. Bro relies on detecting intrusions based on traffic characteristics and content, [16] while Snort relies on rules. Bro has the capability to use Snort’s rules set. Both products are excellent IDS, but for this project, Snort had higher scores in three of the selection criteria: platform compatibility, usability, and support documentation. Snort can be installed on most Linux operating systems, BSD and installed on Microsoft Windows. Bro ran well on BSD but had some compilation errors on a Linux Debian test system, and does not support Windows. Both products were easy to setup but Snort was less complicated to configure and operate. Bro is not a solution for non-Unix environments and it was created with the intention that a UNIX expert would operate it [17]. Snort's logging includes methods to send events to files or databases. Bro only supports sending events to a file. Other methods for storing alerts are left to the administrator. Both web sites had plenty of documentation, but Snort's web site had more examples from the community than Bro's web site [14], [15].

B. AVAILABLE DATABASES FOR STORING IDS SENSOR DATA

The database is one of the most important components for an IDS installation since it will need to handle all of the transactions that take place when events, packets and alerts are logged. There are several open source relational databases available such as the popular databases from MySQL and PostGres. The general requirements section will list some broad requirements before detailed criteria are listed in the Selection Criteria section. .

1. General Requirements

The rate at which the database will need to log transactions has not been determined since stress testing is beyond the scope of this project. As a general requirement, the database should be able to handle moderate transaction loads. The first two selection criteria must be met. The remaining three criteria will use the same ranking system performed in the IDS selection process.

2. Selection Criteria

- Compatible with chosen IDS: Currently Snort is capable of sending alerts to several different types of databases, including MySQL, PostgreSQL, Microsoft SQL Server, Oracle SQL server, and any Unix/Linux Open Database Connectivity (ODBC) compliant database. Only ODBC compliant databases will be considered as this is either a “Yes” or “No” criterion.

- Open source: To avoid license costs, only open source databases will be considered for the project. This is either a “Yes” or “No” criterion.
- Platform compatibility: The database application should support many of the popular operating systems. At a minimum, it should be compatible with the popular Linux operating systems such as Fedora and Debian.

a. Ranking Platform Compatibility

1. Supports minimal complement of common operating systems
2. Supports some common operating systems
3. Supports many common operating systems
4. Supports most common operating systems

- Usability: Some databases come with only Command Line Interfaces (CLI), which mean all administration must be done using CLI commands. To assist with administration tasks, any GUI tools will enhance the product’s usability and make it a selection candidate.

b. Ranking Usability

1. Very difficult to setup and use, has CLI only
2. Moderately difficult to setup and use, has CLI only
3. Little difficulty in setup and use, has CLI only
4. Little to No difficulty in setup and use, has CLI and GUI

- User support and documentation: Database setup and administration can be difficult; therefore, it is imperative that the product have good documentation and support.

c. Ranking Support and Documentation

1. Very little documentation, no user forums or working examples

2. Some documentation, has some user forums, no working examples

3. Good documentation, has user forums, some working examples

4. Good to excellent documentation, active user forums, several clear examples

3. Selection Process

A search for open source databases reveals a plethora of choices. To narrow the selection down to a few candidates only the more widely known databases were considered.. Table 2 summarizes the selection criteria and uses information from a comparison that was performed on five open source databases [18].

Selection Criteria

	ODBC Compliant	Open Source	Platform Compatibility	Usability	Support & Documentation
PostGreSQL	✓	✓	4	4	4
MySQL	✓	✓	4	4	4
MaxDB	✓	✓	4	3	4
Firebird	✓	✓	4	3	4
Ingres	✓	✓	4	3	4

Table 2: Selection Criteria for Database

4. Selection Outcome

The entire list of database products in Table 2 meets the general requirements and scored well in selection criteria, however MySQL and PostGreSQL are the two most popular databases used with Snort [4]. MySQL and PostgreSQL have also been used in past projects therefore; the element of familiarity gives MySQL and PostgreSQL higher scores in usability. Preliminary design work identified a need to use a proxy client to move data from the sensor to the database. MySQL currently has a proxy client that is an alpha version [8], so this extra program may save time and effort with the project. The final choice was to use a MySQL database as the backend storage system.

C. AVAILABLE ANALYSIS TOOLS

Data analysis tools provide a GUI front end to the database that stores the numerous alerts. These tools compute and display statistical information that speeds up the process of sorting through all of the data.

1. General Requirements

Any analysis tool chosen must be able to parse through the Snort alerts. This general requirement narrows the selection of analysis tools down to Snort-compliant tools. The first two selection criteria described below must be met; the remaining three criteria will be ranked similar to the other selection processes.

2. Selection Criteria

- Compatible with chosen database (MySQL) and IDS (Snort): Due to Snort's popularity, there are several analysis tools available to choose from. Some are open source and some are commercial products. Further selection criteria will narrow the list.

- Open source: Only open source products will be considered for selection. Some commercial companies offer free licenses for their products, such as Aanvil [19], but these products are either limited or the code is not available.
- Usability: Analysis tools must provide some basic statistics, sorting and searching capabilities and must display detailed information about the packet. These requirements are necessary to rapidly process large amounts of information.

a. Ranking Usability

1. Provides little or no statistical or packet information, no sorting or searching capabilities

2. Provides some statistical and packet information, has either sorting and searching but not both

3. Provides statistical and packet information, has either sorting and searching capabilities, but not both

4. Provides statistical and packet information, has both sorting and searching capabilities.

- Uses a web interface (web site): Using a web site product allows an analyst to access that product from any workstation that has a web browser. This is not a set criterion, any open source product that installs as a binary will be considered but website products are preferred.
- Extensibility: To display new data in any additional database tables or modified tables may require rewriting some of the code. The programming language in which the product was written will be taken into consideration. Web site products written in popular

programming languages such as PHP or Perl will be considered over web site products that use difficult or obscure programming languages.

3. Selection Process

The selection for an analysis tool is bounded by three criteria, compatibility with MySQL, compatibility with Snort, and it must be open source. Snort documentation lists some open source products that are commonly used as analysis tools [4]. Table 3 lists four products that were examined as possible tools for this project.

	Selection Criteria					
	Snort Compatibility	MySQL Compatibility	Open Source	Usability	Web-based	Extensibility
BASE	✓	✓	✓	4	Yes	✓
SGUIL	✓	✓	✓	4	No	✓
SnortSnarf	✓	✓	✓	2	Yes	✓
SnortALog	✓	✓	✓	2	No	✓

Table 3: Selection Criteria for Analysis Tools

4. Selection Outcome

The most popular tool in the list is the Basic Analysis Security Engine (BASE) that has been an ongoing project for several years [20]. This product is web-based and written in PHP, a popular web scripting language. SGUIL (pronounced sgweel) is another popular product written in tcl/tk [21]. This product is not a web site product and requires communicating with an agent installed on the sensor. Due to the complexity and security considerations of the MYSEA architecture, it is currently not feasible to fit this product into the project. The remaining two products are written in Perl and do not provide any statistical functionality, instead they are used to format and view logs in a more usable display. The choice for this project is to implement BASE as the analysis tool.

D. SUMMARY

Each component required to make a good IDS had suitable products that would work well, but those chosen had essential features that meet the preliminary project requirements. The next chapter will cover the concepts of operation, the formal requirements and design specifications, therefore illustrating how Snort, Mysql, and BASE, come together to make a complete working Intrusion Detection System suitable for the MYSEA architecture.

IV. REQUIREMENTS AND DESIGN

A. INTRODUCTION

This chapter will cover the concept of operations concerning how multiple IDSes could function in a Multilevel Security architecture using a single repository. An analysis of the MySQL protocol will be covered which is essential to understanding the complexity of the problem when dealing with network nodes that rely on a two-way communication protocol such as the MySQL protocol. The requirement section will define how the data from the sensors must be handled to satisfy MLS constraints. The design section will introduce a new architecture that differs from the concept of operations but easily fulfills the requirements section with only minor alterations.

B. CONCEPT OF OPERATION

1. Multiple IDSes using Single Repository

Each Intrusion Detection Sensor monitors the traffic on the network to which it is attached and each network operates at a different classification level such as ‘unclassified’, ‘secret’, and so on. When the IDS sensor sends an alert, the data must be moved from a network at a particular classification level into a database operating at the highest level, i.e., System High (SH). For example, if IDS1, on an unclassified network, sends an alert, the data is sent from an unclassified network to the MLS server’s unclassified port. At the MLS server, the packet is received by an untrusted process labeled U in Figure 5. The untrusted process labeled S in Figure 5 only receives traffic from an IDS attached to the secret network. The process labeled ML in Figure 5 is a trusted process, which can be configured with exemptions to receive data from processes at different classification levels. The database is set at SH, therefore, information flow from the network to the database is allowed since the classification setting of the database is higher than the processes U and S. To view the data, an analyst can use any workstation attached to the MLS network such as the workstation depicted in Figure 5. Since everything is in a SH database the analyst must establish a session at SH before he can view any data. Figure 5 illustrates these concepts in a block diagram format.

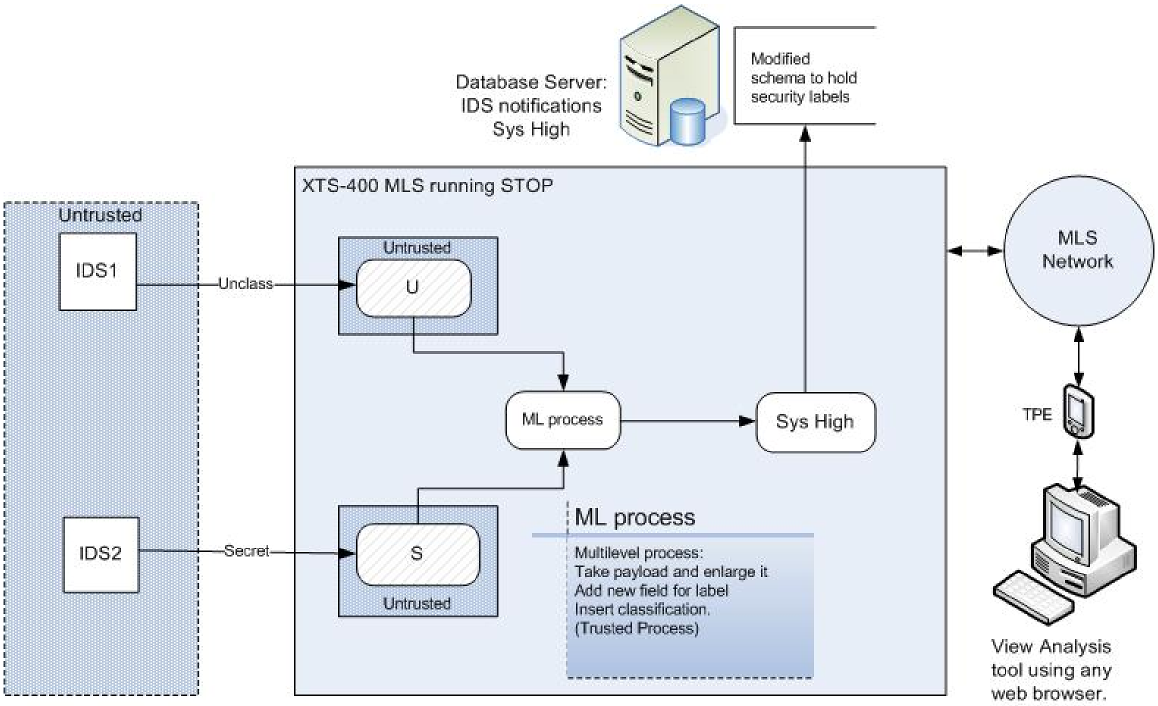


Figure 5: Concept of Operations

The trusted kernel provides logical separation by isolating processes U and S and enforcing system-wide security policies. Processes U and S will act as a listening service (server) and as a sender (client). When the processes receive alerts, the alerts will be forwarded to the ML process. The Sys High process will receive the data from the ML process and forward the information to the database, which is set at SH. The physical placement of each component is illustrated in Figure 6. Each IDS will be placed behind a router with respect to traffic flow. In general, the placement of network components takes into account the inbound and outbound flow of traffic. To illustrate the placement of network components, consider the placement of a router and IDS sensor. If the sensor is placed on the network where it 'sees' the traffic before the router does, this placement is in front of the router. If the router ‘sees’ the traffic before the sensor, then the placement of the sensor is behind the router. The importance of planning the placement of network components is to determine which component will act on the traffic first, the router or the sensor. Because Snort and most other IDSes cannot interpret encrypted data, placing sensors in front of the routers for the SIPRNET network or COALITION network would be impractical because they could not analyze this network traffic.

The database server will serve two functions. One will be as a central repository for storing all alerts. The second function is to host the web application BASE, which will serve as the IDS analysis tool. The analyst will be able to access the BASE web site via an MLS HTTP proxy by establishing a session at the level SH on any COTS client on the MLS LAN.

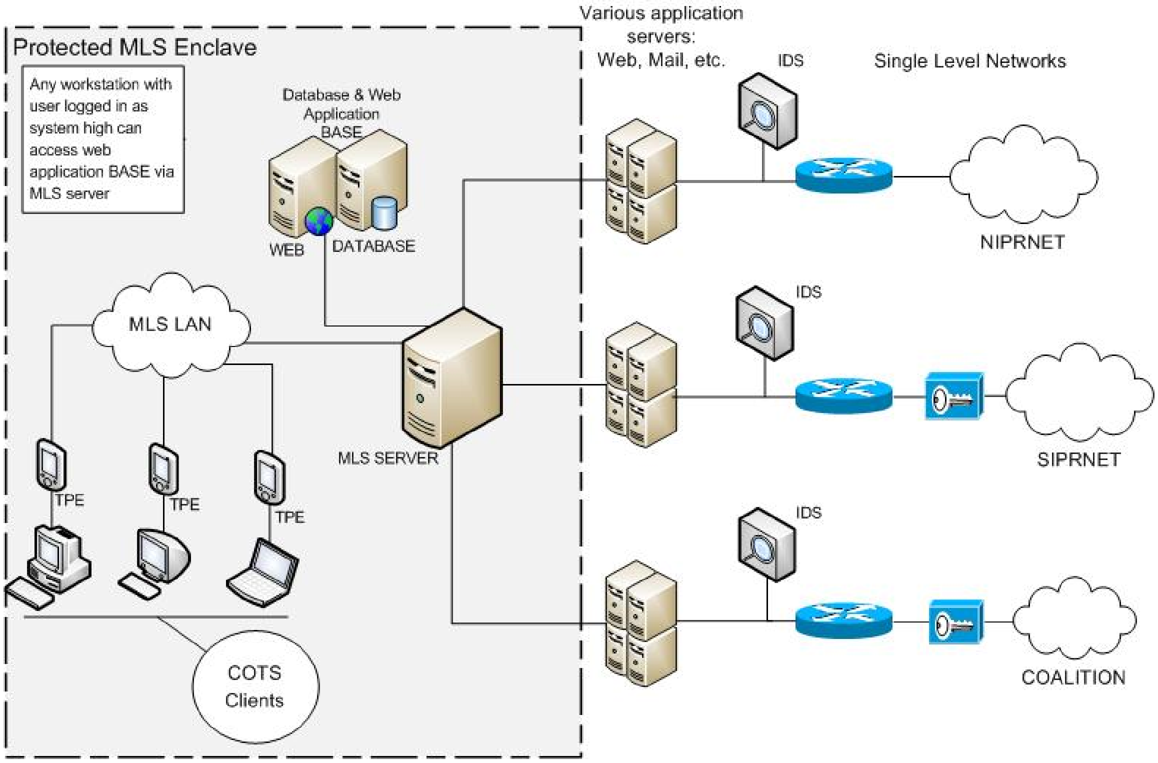

Protected MLS Enclave
Any workstation with user logged in as system high can access web application BASE via MLS server
Database & Web Application BASE
WEB
DATABASE
MLS LAN
MLS SERVER
TPE
TPE
TPE
COTS Clients
Various application servers: Web, Mail, etc.
IDS
Single Level Networks
NIPRNET
IDS
SIPRNET
IDS
COALITION

Figure 6: MYSEA Architecture with IDS

2. MySQL Protocol Analysis

The purpose of this analysis is to illustrate the two-way communication process that takes place between a Snort sensor and the MySQL database. Section 1 discussed abstract processes that dealt with moving data from a single-level sensor to a System High database. To keep the two-way communication process intact for the MYSEA architecture poses a unique challenge. With respect to information flow models, information can flow low to high, however information flow from high to low is an issue because there is a risk of sensitive information leakage. The analysis of the communication protocol between the MySQL database and sensor will give insight into the problem of database to client communications that move data from high to low.

Snort’s alert processes utilize the MySQL client program that is installed on the sensor, to send alerts to a local MySQL database or across a Local Area Network (LAN) to a remote MySQL database. In either case, the MySQL client is responsible for establishing a connection with the MySQL database and acting as a database client for the Snort processes.

The MySQL documentation cited in [23] describes the MySQL protocol but in order to verify how the protocol functions when used by Snort to send data to the MySQL database, a small test environment consisting of one sensor and one database was set up. To capture the exchange of traffic between database and sensor, the alerting mechanism on the sensor was triggered by performing a port scan on the sensor. The alert data sent by Snort and reciprocal response data sent by the MySQL database was captured and analyzed. The discussion that follows is a compilation of observations made during the experiment coupled with MySQL documentation.

When the Snort process is initialized, a function calls the locally installed MySQL client to establish a connection with the database that is listed in the Snort configuration file: snort.conf. In response to this connection request, the database sends a ‘server greeting’ message containing attributes about the

MySQL server such as version number, server capabilities and server status. The MySQL client (on the sensor) responds to the greeting message with a login request, passing the credentials for a username that has already been configured in the Snort database resident on the MySQL server and stored in the Snort configuration file resident on the sensor. If the credentials are validated, the server responds with an OK message.

After the MySQL client receives the OK message from the MySQL server in response to the authentication of credentials, Snort queries the database for an existing 'sid' (sensor identification) and 'hostname' to either establish a new identification if an old one does not exist, or to continue from the previously established session using the existing identification.

The series of steps discussed, from connection request by the MySQL client to establishing sensor identification, is used to generate a new persistent session between sensor and database. The session between the MySQL client on the sensor and the MySQL database remains open until the Snort process is terminated or communications between the MySQL database and MySQL client are interrupted. The reason for leaving the session open is to reduce the overhead of establishing a new session every time an alert is sent to the database.

Before sending an alert Snort will check for duplicate data by issuing a SQL SELECT command that queries the Snort database looking for matching alert signatures. If the alert signature already exists, then the event counter in the EVENT table of the Snort database is incremented and only new information such as source and destination IP addresses and the contents of the packet's payload are sent to the database. If the alert signature is new, then new rows are created for the alert, including a new row in the SIGNATURE table. Information that is sent to the database in an INSERT command includes the sensor identification, time stamp, source and destination IP addresses, the signature that defines the attack, for example 'Portscan' and the payload of the captured network traffic that caused the alert. The final command sent by the MySQL client is a COMMIT command that instructs the database to write the new data to the database. Every command sent by the MySQL client, such as BEGIN, INSERT, or COMMIT, the server responds with one of the following: OK, ERROR, or a Result Set in response to a SELECT command. The sequences provided in this discussion illustrate the two-way communication protocol used by the MySQL server and MySQL client.

Responses from the MySQL server to the MySQL client on the sensor move from high to low, therefore, information in the return traffic will require downgrading by a trusted process before sending the response out to the sensor. Downgrading of classified documents is normally performed by a guard, which may also sanitize the contents of the document [22]. A sanitization process could be implemented to verify that return traffic contains only valid responses. Responses such as OK or ERROR are trivial to verify by inspecting the payload of the datagram. Responses to the SELECT command that contain data sets will be more complicated because of the myriad possibilities of legitimate responses coupled with the possibility that the payload could contain sensitive information.

Alert information stored in the database requires a security label. If the data is not labeled then an analyst could accidently include sensitive data in an unclassified report. The following is an example of a row of data from the Snort database EVENTS table:

1, 29, 1, ‘2008-04-10 19:24:41’

sid, cid, signature, timestamp

The columns, delimited by a comma in this example, are the sensor identification (sid), a counter (cid), an index to the signature table (signature), and a timestamp (date). If this information was sent from IDS2 on the secret network showed in Figure 6 then the information should be labeled ‘SECRET’. To accomplish this labeling the table requires another column for the label and the INSERT command sent by the MySQL client needs to be expanded to apply the label. The following is an example of a row of data after the modifications:

1, 29, 1, ‘2008-04-10 19:24:41’, SECRET

sid, cid, signature, timestamp, class_label

The task of expanding the INSERT command and labeling the data must be performed by a trusted process on the MLS server to ensure the veracity of the labels. A critical assumption is made that the database is benign and does not alter the labels, otherwise the information cannot be trusted and the architecture is flawed.

C. REQUIREMENTS

This section will cover the requirements to construct a working prototype based on the issues disclosed in the concept of operations from section A and Section B.

1. IDS

The sensor chosen for this project is an open source product called Snort. The reason for selecting Snort was covered in Chapter III. One of the goals of this project is to establish a template for implementing an IDS, therefore the only modifications to Snort will be those required to make the product operational. For example, the configuration file /etc/snort/snort.conf requires editing to set up certain parameters to make the IDS functional. Some of the required parameters are items such as what network to monitor and the authentication parameters for authenticating to the database. These types of modifications will be allowed. The implementation section will cover setting up the configuration file.

2. Information Flow Control

The communication flow between the database and sensor must not violate MLS policies. Any information flowing from high to low would require a guard process to verify that no classified information is leaked.

3. Processes Running on MLS Server

There are two requirements for running new processes on the MLS server. First, if the process is to be granted security and integrity exemptions the process must be a trusted process. A trusted process is a program, which is granted elevated privileges such as security and integrity exemptions. The behavior of the process must be analyzed to verify that it does not violate the overall intent of the system security policy. An untrusted process does not require such an analysis; therefore, behavior of the process is unknown but may be required to satisfy the system’s operational functionality. Second, any new code introduced must be compatible with the STOP 6.3 operating system.

4. System High Database

The chosen database must be able to run on an operating system that will allow it to run as a system high database in the MLS architecture. Snort supports various ODBC compliant databases.

D. DESIGN

This section covers the design process based on the concept of operations and requirements sections. Preliminary tests using a small Linux environment were used to validate some concepts to help with the design process. Results from these tests are included in this section.

1. Preliminary Tests

A small network consisting of three Debian 4.0 Linux systems was configured to prove some concepts that were described in the concept of operations. One system was configured as a sensor, one as the database server and the last one was a simulated XTS system, which did not emulate any MLS capabilities.

To assist with moving traffic between processes U, S, ML, Sys High, and the System High database (see Figure 5) the MySQL Proxy program was researched and tested as a possible solution. A proxy server services the requests of its clients by forwarding requests to other servers. A proxy service can act as client by sending requests and as a server by accepting requests. The MySQL Proxy program performs similarly to a standard proxy program however; the MySQL Proxy program only handles communications between MySQL clients and MySQL databases using the MySQL protocol.

The first test was to install one proxy on a Debian 4.0 Linux system simulating an XTS and observe alerts transitioning from the IDS sensor to the database via the proxy. Figure 7 shows the network layout for the first test. Results of the test were satisfactory; therefore, the second test was performed.

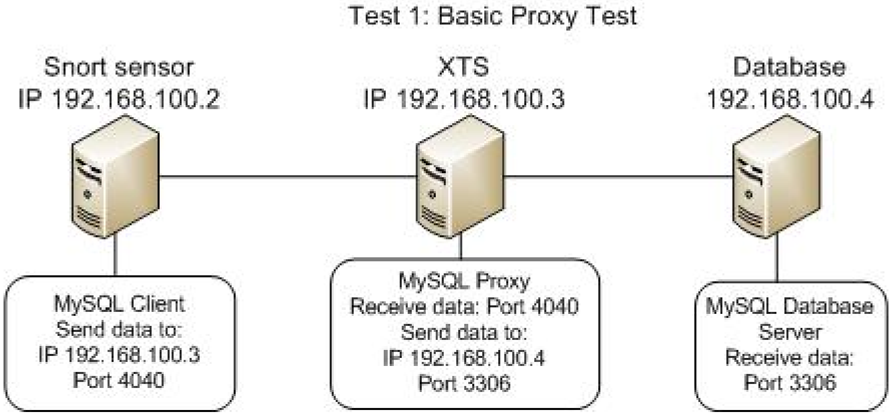


Figure 7: Test1 Basic Proxy Test

Moving data from process U to process Sys High would require using multiple MySQL proxies therefore the second test experiments with using the proxies in a daisy chain configuration. Figure 8 illustrates the setup of two daisy-chained proxies.

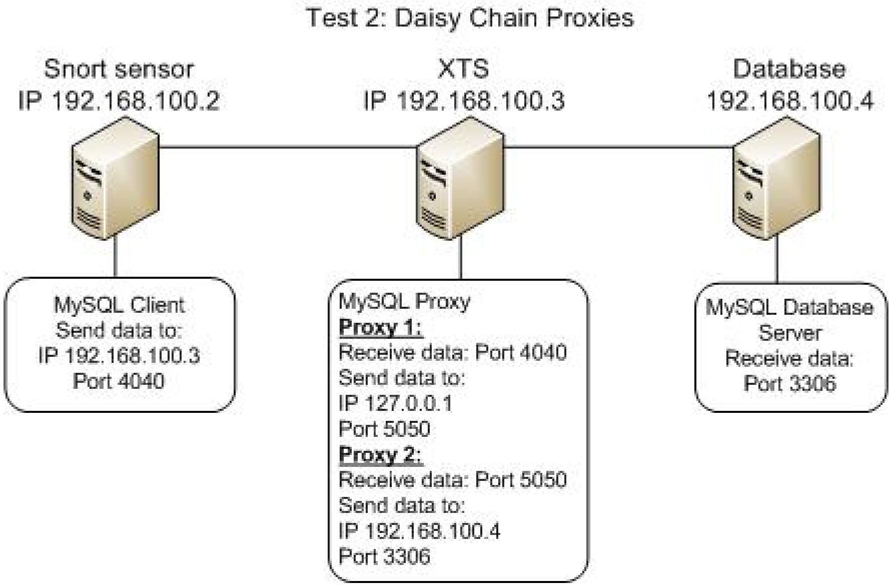


Figure 8: Daisy Chain Proxies

To create the daisy chain between the two proxies requires Proxy 1 to be configured to send data to Proxy 2, which has been configured to use the local host address 127.0.0.1 and port number 5050. Proxy 2 then sends the data to the external database server at IP address 192.168.100.4. The second test was successful and demonstrated data moving from the sensor to the database via the two proxies.

The third test utilizes MySQL Proxy's built in Lua interpreter, which has the capability to monitor and change the contents of MySQL packets. Lua is another interpreted scripting language, similar to Perl or PHP [10]. Lua can be installed as a separate program but the developers of MySQL Proxy compiled the interpreter into the MySQL Proxy program [26]. Adding Lua to the MySQL Proxy program gives developers the ability to write scripts that can modify or monitor SQL statements as the data stream flows between client and server. For this test, a Lua script was written to perform a simple SQL injection by adding another INSERT command to append a classification label to the data being inserted. Figure 9 shows the layout for the third test.

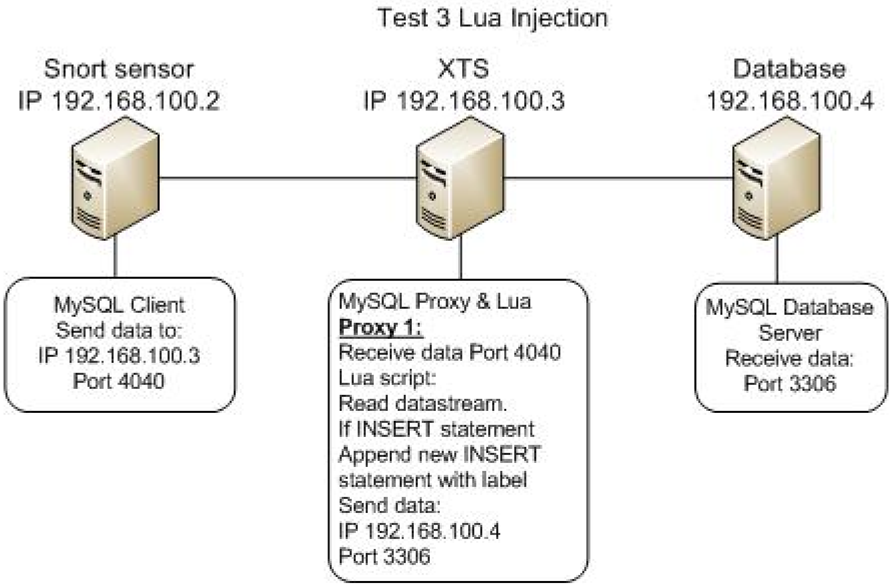


Figure 9: Test 3 Lua Injection

This test also required a small modification to the Snort database schema to include the additional data field for the label. The results from the third test were satisfactory. This series of tests demonstrated that using one or more MySQL proxies to move data between the required processes U, S, ML, Sys High and SH database appears to be a feasible solution.

One constraint on any possible solution is porting new code to run on the STOP 6.3 operating system. This OS is similar to Red Hat 8 in regards to the libraries supported and the kernel programming interface. To determine if a new program might compile on the STOP 6.3 OS it is first compiled on a Red Hat 8 system. The caveat however is that any compilation must use only the base Red Hat 8 image. This means there cannot be any foundational upgrades performed to the Red Hat 8 operating system such as upgrading kernel code because this would move the version level away from the original level.

The fourth test was to compile the MySQL proxy code on a base Red Hat 8 system. To complete the configuration phase of the compilation process a few libraries were installed such as glib-2.6.0, libevent-1.3e, lua-5.1.3 and the glibc package was upgraded from 2.2.9 to 2.3.2, which was the last version of glibc configured for Red Hat 8. Compiling MySQL proxy on Red Hat 8 failed.

Compilation of the MySQL proxy requires MySQL client and development files greater than version 5.0, which is not supported on Red Hat 8. The glibc 2.4 package is also required but this package requires kernel 2.6.9. When executing 'make' the first compilation error appears in mysql-proxy.c at line 194 with the error message, “structure has no member named org_name.” Since there were only five errors in this file of this type, the errors were resolved by commenting out the offending lines of code. The code was not vital because it only printed the names of the authors and their contact information. Re-running 'make' compiles mysql-proxy.c but yields new compilation errors in the file network-mysqld.c. Four variables were not found probably due to references to an incorrect version of a header file. The same source code was successfully compiled on Debian 4.0 by installing the following packages:

pkg-config, liblua5.1-0, liblua5.1-dev, libevent-dev,(from .deb package),

libevent1, libglib2.0-0, libglib2.0-dev, libmysqlclient-dev, libncurses5-dev_5.5-5_i386.deb, libreadline5-dev_5.2-2_i386.deb

Due to the compile errors, kernel and library requirements, and the fact that the code is only in the alpha stage of development, a determination was made that it was not feasible to try to alter the code to make it conform to Red Hat 8.

Despite the fact that the Proxy code performed well on the Debian 4.0 systems, and the fact that it cannot run on Red Hat 8 means there is a high probability it will not run on the STOP 6.3 operating system. Another issue that must be considered is the trustworthiness of the code. The proxy code consists of 19 files and over 12000 lines of code, none of which has been examined to determine if the code will always perform correctly. The level of the code is only in the alpha stage of design, which presents another concern for using this code.

These two factors prompted a change in the design strategy that will differ from what was covered in the concept of operations.

2. New Design for Implementation

The first change from the original concept of operations was to switch the database software from MySQL to PostgreSQL. The primary reason for the change is that the new STOP 7 version comes with PostgreSQL 8.1 installed, however this version level of PostgreSQL is too high for STOP 6.3 therefore the PostgreSQL version will be 7.4.18, a version compatible with Red Hat 8. The Snort product includes a SQL script for creating the schema required for the Snort database for both MySQL and PostgreSQL and is fully compatible with the Snort sensor. The analysis tool BASE is also configured with full PostgreSQL support. Taking into account that future versions of STOP will support PostgreSQL along with the PostgreSQL compatibility with the IDS components, switching database software will provide a more stable implementation for this project.

Another major change is the elimination of the external database server and external web server. These components will be hosted on the XTS400 and will be constrained by MAC policies enforced by MYSEA trusted processes and STOP OS itself. The IDS will communicate directly with the databases instead of communicating through a proxy. Even with PostgreSQL, there is still a requirement to provide two-way communication between the database and the sensor’s database client. Figure 10 illustrates the new design that satisfies the requirement to use multiple IDS sensors in a MLS architecture.

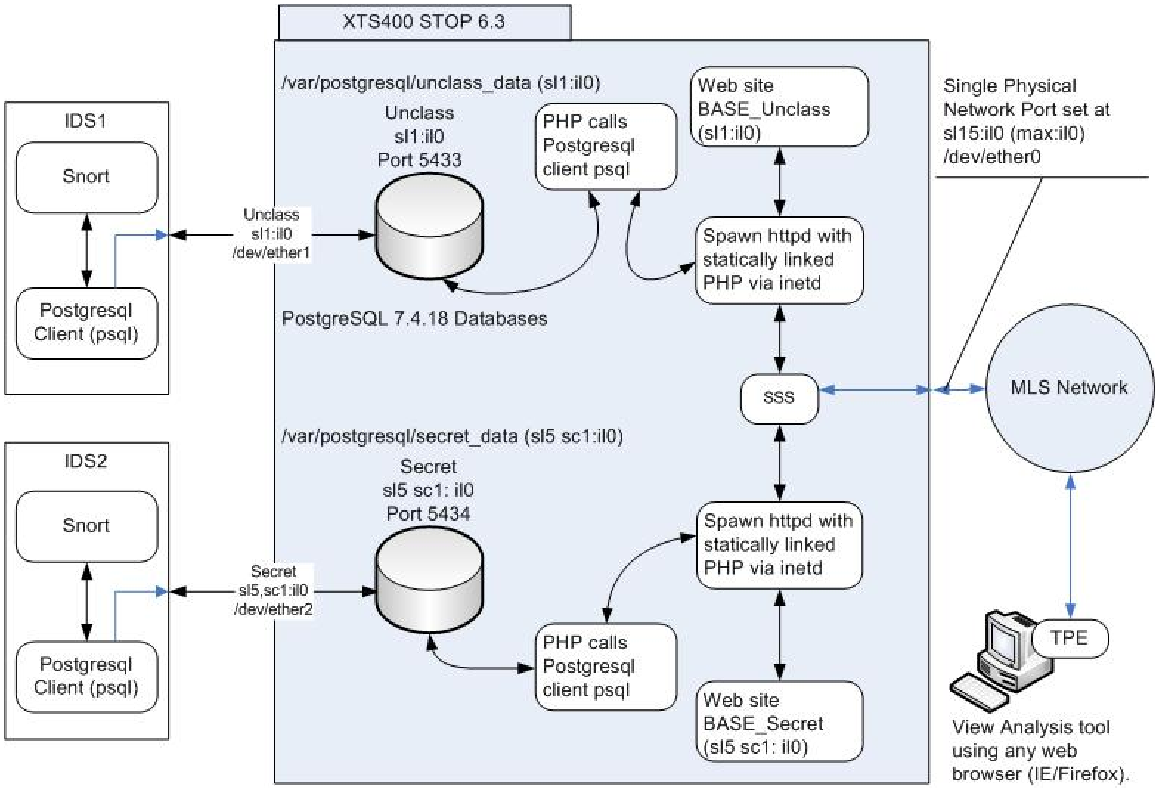


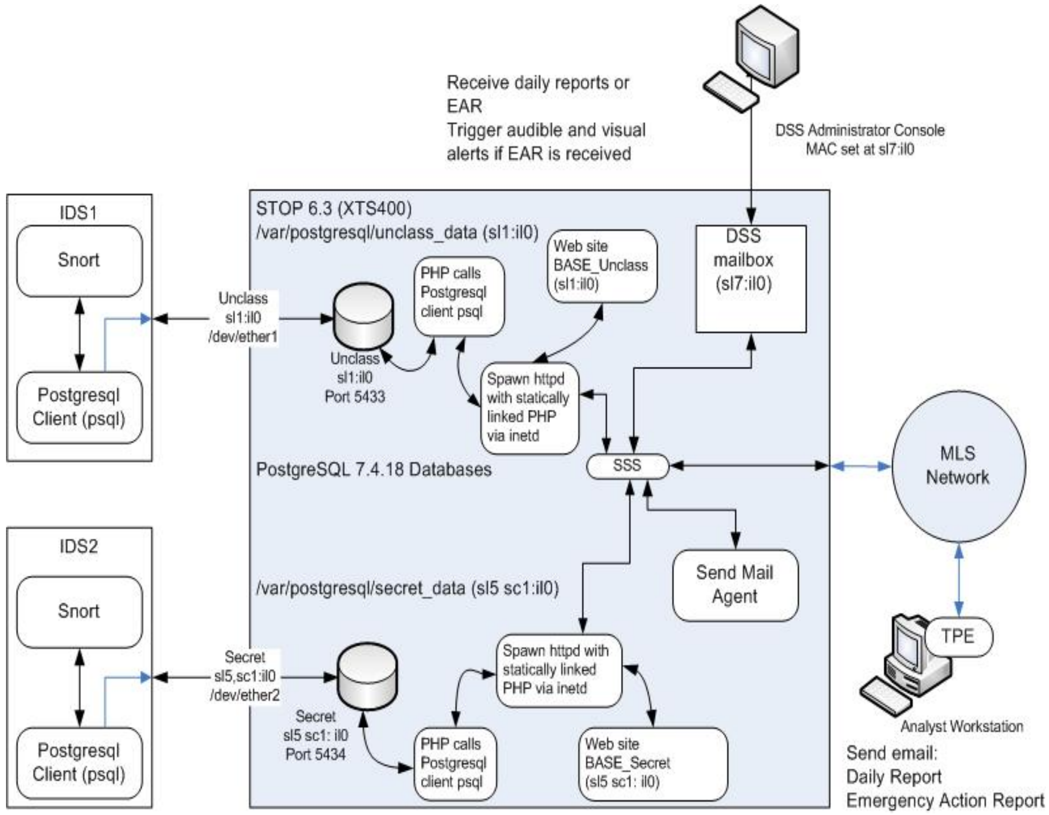
Figure 10: IDS Implementation Design

When a Snort sensor either is initialized or sends an alert it will invoke the PostgreSQL client that is installed on the sensor. Reconfiguration of the network ports on the XTS400 to match the security and integrity level of each PostgreSQL instance will allow the IDS to communicate with their respective PostgreSQL databases. There are databases at each classification level because there is no trusted component to label or fuse the data as it is received. This design requires two instances of PostgreSQL and two separate BASE web applications.

Access to the BASE web application is through the Apache web server application ‘httpd’ that is spawned by the SSS when an analyst makes an HTTP request. All data queries are invoked by the analyst using the PHP web pages that comprise the BASE application. In turn, the PHP interpreter makes a call to the PostgreSQL client to execute the requested SQL query and returns the result set to be rendered in the PHP pages as readable data for the analyst. Even though the alert data are separated, the analyst has an expedient access method to the data using web browsers. However, the analyst will have to perform manual correlation of data by generating reports and comparing results.

3. Integration with DSS

Figure 11 is a conceptual design that illustrates the use of email to notify the DSS administrator with IDS reports. The actual implementation of this approach is outside the scope of this thesis but the design is presented to demonstrate how an analyst could report the status of network activity to the DSS administrator.



Receive daily reports or EAR
Trigger audible and visual alerts if EAR is received
DSS Administrator Console
MAC set at sl7:il0
IDS1
Snort
Postgresql Client (psql)
STOP 6.3 (XTS400)
/var/postgresql/unclass_data (sl1:il0)
Unclass
sl1:il0
/dev/ether1
Unclass
sl1:il0
Port 5433
PHP calls Postgresql client psql
Web site BASE_Unclass (sl1:il0)
DSS mailbox (sl7:il0)
Spawn httpd with statically linked PHP via inetd
SSS
PostgreSQL 7.4.18 Databases
MLS Network
IDS2
Snort
Postgresql Client (psql)
/var/postgresql/secret_data (sl5 sc1:il0)
Send Mail Agent
Secret
sl5,sc1:il0
/dev/ether2
Secret
sl5 sc1: il0
Port 5434
Spawn httpd with statically linked PHP via inetd
PHP calls Postgresql client psql
Web site BASE_Secret (sl5 sc1: il0)
TPE
Analyst Workstation
Send email:
Daily Report
Emergency Action Report

Figure 11: IDS Integration with DSS

The MYSEA server already supports the open source email program ‘send mail’ to provide email capability to the users on the MLS network. Since the analyst is working from a COTS client, he could generate IDS reports using information from the BASE application and send those reports to the DSS administrator via email. Reports sent to the DSS administrator would be categorized based on urgency. A string of text placed in the subject line of the email would identify the urgency level such as “Daily Status Report” for routine operations or “Emergency Action Report” for a critical alert that requires a response. Critical alerts would also trigger an alarm on the DSS administrator console.

In the current MYSEA implementation, the DSS administrator can only access email from a COTS workstation on the MLS network, not from the system console. Further investigation is needed to determine the changes to the current MYSEA code to allow the DSS administrator to receive IDS reports via email at the console.

E. SUMMARY

This chapter covered an analysis of the original design concept that resulted in alterations to the design due to unexpected ‘cutting-edge’ software conflicts with the older STOP system. The new design incorporates a few compromises such as splitting out the database repositories, but overall provides a viable functional prototype that can be fully incorporated into the MYSEA environment after the PostgreSQL processes are ported to use the MYSEA socket interface as explained in the future work section in Chapter VI.

The next chapter, Implementation and Testing, will cover the installation and configuration of these components in two different environments: that of Red Hat 8 Linux, and for the XTS400. After the IDS components are installed, the final part will cover functional testing.

V. IMPLEMENTATION AND TESTING

A. INTRODUCTION

This chapter will review the implementation of the IDS components: sensors, database, and analysis tool, on the Linux Red Hat 8 system and describe the changes required to integrate all of these IDS components within the MYSEA architecture. The analysis section documents problems encountered during the implementation phase for the MYSEA architecture. The chapter concludes with a functional test plan that was designed and used to exercise the entire suite of components to verify that all components function as a coherent Intrusion Detection System. Appendix A and Appendix B provide detailed instructions for implementing these IDS components.

B IDS IMPLEMENTATION ON THE RED HAT 8 SYSTEM

This section covers the standard implementation of the IDS components on the Red Hat 8 Linux operating system. The reason for using the Red Hat 8 system is covered in Chapter IV, and is to use an OS that is compatible with the STOP 6.3 operating system with respect to the version of the operating system's kernel and supported libraries. The two main components that will be installed on Red Hat 8 are the PostgreSQL database and the BASE web application. The Snort sensor was, installed on a separate machine.

1. IDS Sensor Installation (Snort)

The detailed installation of Snort on the Debian 4.0 operating system is covered in Appendix A and Appendix B. The Debian operating system was installed with the minimal libraries and programs required to create a functional Linux operating system, which means there were no graphical user interfaces (GUIs), or any other gratuitous services that could present attack vectors. After the OS was installed, other required software programs were added such as the

GCC compiler, PostgreSQL client and the libpcap library, which is a library required to support Snort’s packet capturing functions.

Snort version 2.8.0.1 was installed using the source code obtained from www.snort.org. This was accomplished using the standard Linux installation method of running the ‘configure’ script then the ‘Makefile’ script and concluding with ‘make install’, which places the binary ‘snort’ into the ‘/usr/local/bin’ directory. Completing the Snort installation required editing the Snort configuration file ‘snort.conf’ to enable parameters that determine items such as what network space to monitor, where to send alerts, and the location of the directory containing Snort rules. As a security measure, the open source firewall ‘lokkit’ was installed and configured to allow only the required ports for communication between PostgreSQL client and PostgreSQL server.

2. PostgreSQL Server Installation

The installation of PostgreSQL 7.4.18 server used source code obtained from www.postgres.org. The installation followed the standard Linux installation method of running the ‘configure’ script, followed by ‘Makefile’ script, and concluding with ‘make install’, which installed the binaries to ‘/usr/local/pgsq’ and other subdirectories under the ‘pgsql’ directory. Before the PostgreSQL server could be started, a database cluster was created by running the initialization program called ‘initdb’ with a directory location as an argument. A database cluster is a collection of databases that are managed by a single server instance [33].

To satisfy the design requirement to use separately managed databases for storing alerts, this implementation required two instances of PostgreSQL server: one instance for an unclassified Snort database, and another instance for the Secret snort database. Creating additional PostgreSQL instances required initializing additional database clusters. To simulate the PostgreSQL implementation that will be required on the STOP 6.3 OS, two database clusters were created: ‘/var/postgresql/unclass_data’, and ‘/var/postgresql/secret_data’.

Each database cluster contained two configuration files that required editing. The first file, ‘postgresql.conf’, is used to establish various network parameters such as those to allow TCP/IP connections and to set the port number for TCP/IP communications. Each database instance used a distinct port number. The unclassified database used port 5433 and the secret database used port 5434. The default port for PostgreSQL is 5432.The second file, ‘pg_hba.conf’, is used to set the client authentication method for local and remote connections.

The pg_hba.conf contains a synopsis of the authentication methods, which are also covered in the PostgreSQL documentation. Authentication for this lab experiment used the MD5 method. This method compares an MD5 hash of the stored password in the PostgreSQL pg_shadow catalog [32] and the hashed password that is offered by the PostgreSQL client installed on the sensor. Issues with MD5 and other authentication topics are covered in section C. To create databases for the Snort sensors required running a script called ‘createdb’, however, creating the database was performed by the intended owner of the database to ensure proper permissions. For this reason a Red Hat 8 system user called ‘snort’ and a PostgreSQL database user called ‘snort’ were created just for the purpose of creating a database called ‘snort’. The schema for the database was generated by running a SQL script included with the Snort distribution. After the schema was created, the databases were ready to receive alerts from their respective sensors.

3. Web Application BASE Installation

The Basic Analysis Security Engine (BASE) is a web application written in PHP with cascading style sheets to provide the web application with some uniform styling. BASE takes advantage of a database abstraction library called ADOdb, written in PHP, which is designed to increase performance between the database and web application [27]. Both of these products use the PHP scripting language interpreter, which needs to be installed and configured to work with the Apache web server.

There are three methods for installing the PHP interpreter to function with the Apache web server. One method will statically embed the PHP interpreter into the Apache binary (httpd), which is the fastest way to run PHP because both the PHP binary and httpd binary load into memory as one application. The httpd binary will increase in size by as much as five times its original size depending on the PHP options that were installed at compile time. The second method compiles PHP as a Dynamic Shared Object (DSO) module, which provides a way to build PHP as a separate binary, but links it to the httpd binary to load at run-time into the memory space of the httpd program. The final method uses PHP as a Common Gateway Interface (CGI) binary. This method does not integrate into the Apache executable file httpd but does require specific configuration settings in order to work securely and correctly.

The DSO method was chosen for the Red Hat environment based on modularity considerations. The main advantage in using the DSO method is that if PHP requires recompilation to add additional features or upgrades, only the PHP binary is recompiled, not the httpd binary. When the static method is used, upgrading PHP requires recompilation of the PHP binary and the httpd binary.

After compiling Apache to use DSO modules and installing PHP, the BASE web sites were setup under the default Apache root directory called ‘htdocs’ that is used for serving web pages. Two BASE web sites were configured: one to access the unclassified PostgreSQL database and another to access the secret PostgreSQL database.

To finish the BASE installation required accessing the BASE web site’s setup.php web page using a web browser from either the local server or any host that can access the Red Hat 8 server. This setup page is used to fill in parameters that will be written to a configuration file called base_conf.php.The setup is only performed once to initialize these parameters and create six new tables in the Snort database. After the setup is complete, the login page was displayed for the BASE web site. After logging in using the user name snort, the home page displayed a plethora of options for viewing and displaying alerts.

C. IDS IMPLEMENTATION FOR THE MYSEA ARCHITECTURE

This section covers installation of the three IDS components in the MYSEA architecture. The database software PostgreSQL and web application BASE were installed on the STOP 6.3 operating system, which is the base for the MLS server. The MLS server manages objects by enforcing MAC policies on every object, e.g., users, files, directories, and processes. Users can change their security and integrity level by invoking the Secure Attention Key SAK (Alt+ SysReq), and then entering a security level and an integrity level. When working with objects the level of the object must be taken into account. A good rule of thumb when dealing with objects is knowing the Bell-LaPadula model for security (no write down, no read up) and Biba model for Integrity (no read down, no write up).

The STOP 6.3 OS employs several levels and categories for security and integrity which gives developers flexibility in creating various classification levels. This installation is only concerned with the following five classification levels:

- min:oss - Defined as security level 0 and integrity level 3 with all integrity categories enabled. The main directory structure of the OS is set at this level.
- min:il3 - Defined by the MYSEA developers as SysLo (System Low), this is the setting for the MYSEA directory structure /usr/local/MYSEA and MYSEA web directories /home/http.
- sl1:il0 - Defined as security level 1, integrity level 0, and is also defined as SIM_UNCLASSIFIED in the MYSEA architecture. This setting denotes the classification level ‘simulated unclassified’ for any object or session that is set to this level.

- sl5 sc1:il0 - Defined as security level 5 with category 1 set, and integrity level 0. This level serves as the SIM_SECRET level to denote the classification level ‘simulated secret’ for any object or session that is set to this level.
- sl15 sc0…sc63:il0 - Defined as security level 15 with all security categories set, and integrity level 0. This level serves as the MYSEA system high level.

1. IDS Sensor Installation (Snort)

The implementation for Snort was the same as the installation discussed in Section A; therefore, the sensors are reused for the MYSEA architecture. Only the configuration file ‘snort.conf’ required changes to reflect different parameters such as database IP address or port numbers.

2. XTS400 Network Settings

The XTS400 server used for this implementation had four physical network ports. One port is used for the MYSEA MLS network and two ports are used to simulate two single-level networks: unclassified and secret. The security setting for the MLS port was set to the maximum level. The reason for the maximum setting was to prevent application processes such as Apache direct access to the NIC. Users establish a classification session using the TPE installed on the client. Sessions are managed by the Secure Session Server (SSS) to allow for all possible session levels.

Currently there is no MLS routing or session server for hosts connected to the single level network ports. To allow TCP/IP communications between the sensors and their respective databases the security level of two network ports were reconfigured to match the security level of the two PostgreSQL instances. Ethernet device one (/dev/ether1) is set to sl1:il0 (SIM_UNCLASSIFIED) and

Ethernet device two (/dev/ether2) is set to sl5 sc1:il0 (SIM_SECRET). This allowed the PostgreSQL instances to offer database services running at these two different classification levels.

3. PostgreSQL Installation

The installation procedure for the PostgreSQL server on the STOP 6.3 OS is similar to those for Red Hat 8 and were performed as the user ‘admin’ at level min:oss, which means the newly installed directories and binaries are at the level min:oss. The database directories /var/postgresql/unclass_data and /var/postgresql/secret_data, required for the database clusters, were created using STOP’s file manager program, to set MAC and DAC permissions on these directories. The unclass_data directory is set to sl1:il0 and the secret_data directory is set to sl5 sc1:il0.

The database initialization on the Red Hat 8 system could not be performed using the system account *root* because that method is not allowed by the PostgreSQL process, therefore a non-privileged account was created to initialize database areas and run the PostgreSQL process. The STOP operating system does not support the concept of a root user, but instead of having the default account *admin* initialize the databases and run the PostgreSQL processes a lesser privileged account called 'postgres' was created using STOP's account management program.

The remainder of the installation followed the same steps as the Red Hat 8 installation. The configuration files had to be edited, the snort user and snort database must be created. Performing these steps required being at the applicable classification levels.

To avoid having to start the databases manually, two daemons were created using STOP’s daemon editing program. When the databases were completed, a quick test using the locally installed PostgreSQL client ‘psql’ was used to ensure connectivity and that the databases were ready to receive data.

4. Basic Analysis Security Engine (BASE) Installation

Implementing PHP on STOP 6.3 as a DSO module, which was the method performed in the Red Hat 8 environment, was not possible due to modifications performed on the Apache source code. Apache was ported to the MYSEA environment to work in conjunction with other MYSEA components. To compile the DSO module PHP uses a support program from Apache called ‘apxs’, which simplifies the creation of DSO files for Apache modules [28]. To compile PHP as a DSO module Apache must have been compiled with the module ‘mod_so’ enabled. This is checked by the PHP configuration script via the apxs support program. To manually check to see which modules are installed the httpd program is executed with the ‘-l’ option, for example, enter ‘./httpd -l’ at the command prompt, press the enter key then check the output. One line of the output will contain the string mod_so.c, which indicates Apache was compiled with the dependent module mod_so. The apxs script performs this same procedure programmatically then performs a string match on the output. If the string mod_so.c is not present, the script fails outputting an error that Apache was not compiled with mod_so enabled.

Performing this check on the altered httpd binary file yielded no visible output. Without the proper output, the apxs support program fails, which in turn causes the PHP configuration script to fail. To circumvent this issue PHP was compiled as a static module. A static compilation of PHP embeds the PHP interpreter into the httpd binary program. Both of these methods, static link and DSO, have positive and negative aspects that will be examined in the next section.

Two directories were created to support the BASE web application (/idsdemo/base_unclass, /idsdemo/base_secret) under the Apache root directory /home/http/htdocs. Using STOP’s file management program the proper security and integrity levels are applied to the separate directories. Completing the installation required using a web browser on a client machine that has the TPE program installed. First, a user session was set to ‘SIM_UNCLASSIFIED’ and then the URL for the BASE setup page for the base_unclass web site was entered into a web browser. This web page instructs the user to provide various parameters then finalizes the installation. The user session was changed to SIM_SECRET and then the URL for the base_secret site was entered into a web browser to perform this same procedure at this level. When both sites are configured, there are two identical web applications with one application for the unclassified data and one application for the secret data.

D. ANALYSIS OF MYSEA IDS IMPLEMENTATION

This section will conduct an analysis of issues that were identified during the implementation of the IDS components in the MYSEA architecture. These issues are deferred for future work but are presented in this section to provide some insight. The main issues to be covered are sensor authentication, PHP access to PostgreSQL and PHP integration with Apache.

1. IDS Sensor Authentication

PostgreSQL server provides several authentication methods from the simple unsecure ‘trust’ authentication method, which requires no password, to the highly secure Kerberos authentication method. This implementation used the MD5 encrypted password method as a middle ground solution. Using Kerberos requires an authentication server, which the MYSEA architecture currently does not employ. Due to the weakness of the MD5 protocol and because the way in which it was implemented in PostgreSQL, MD5 should not be a long-term solution.

The MD5 authentication methodology stores a hash of the user’s password in the pg_shadow table, which is part of the system catalog and is not accessible by public accounts [32]. When the hash is generated, the user’s name is used as the salt vice using a randomly generated salt [29]. Most PostgreSQL implementations tend to use the username ‘postgres’ as the account name during initialization of database clusters and as the owner of the PostgreSQL processes. Snort implementations tend to use the word ‘snort’ as the owner of the snort database. Any hacker who compromised a sensor or realized there are active communications between a Snort sensor and a PostgreSQL database could possibly have two salts, which would allow for easier breakage of the hash once it was retrieved.

Another security issue with IDS sensor authentication is the storage of passwords in the snort.conf configuration file. One section of the snort.conf file stores database connection parameters such as IP address, port number, username and password. If a hacker gains access to a sensor as root, they now know one password that will permit access to the remote Snort database. There is currently no real solution for this security hole.

2. PostgreSQL, BASE and Multilevel Access

Accessing a PostgreSQL database is performed by using client software that utilizes the PostgreSQL protocol. PostgreSQL software installation includes a client named ‘psql’, which is normally used for accessing the databases. Making a connection consists of invoking the program and passing in a user name and password as minimum arguments. All authorized accesses to a particular database are treated as read/write access unless specific permissions have been established at the table level.

The BASE web application uses the resident ‘psql’ client installed on the STOP 6.3 system to service SQL queries embedded in the PHP scripts that comprise the BASE application. When an analyst establishes a SIM_SECRET session, that analyst has read/write access to the base_secret web site but also has read access to the base_unclass web site. If the BASE authentication method (Step 3 of 5 from the BASE setup page from Appendix A and Appendix B) is being used then the first page of either BASE web site is a login page that validates given credentials based on a set of established credentials stored in the Snort database. When the analyst enters credentials for the base_unclassified web site, the return will be a PostgreSQL database connection error. The reason for the error is the enforcement of the MAC policies. This behavior can be duplicated locally on the XTS400 by logging in as ‘postgres’, setting the security and integrity levels to <sl5 sc1:il0>, and attempting a login to the unclassified database using the ‘psql’ database client program. The output error will be identical to that viewed in the web page.

To provide true MLS support an analyst should be able to view the contents of a database in read only mode that is at a lower security level and equal or higher integrity level than the established session level. Currently analysts will only be able to view data whose level equals the established session level.

3. PHP Integration with MYSEA Apache

A standard Apache installation is configured to run the httpd process under a parent process, which forks a number of child processes to handle incoming HTTP requests. Among the resources for this configuration are memory space for the parent process and memory space to handle new connections. When PHP is integrated into Apache the memory usage increases based on the request of dynamic content, i.e., PHP web pages. PHP is allocated memory using the malloc (memory allocation) function and uses the allocated memory until the PHP process is terminated.

The MYSEA implementation of Apache utilizes the same approach used by inetd daemon to handle HTTP requests. When a HTTP request is received from a client, the SSS process, an inetd-like program, launches the httpd server program to handle the request and then terminates the program when the request is complete. With this method, each HTTP request from a client causes a new httpd program to spawn and run as the user account making the request instead of using a non-privileged account such as ‘apache’. This method is suitable for the MYSEA architecture since the application processes are not trusted and should not be able to directly access the MLS network. Before implementing PHP, the MYSEA httpd file size was 1.1 Megabytes, after the PHP implementation the file size grew to 5.8 Megabytes. When a client makes a HTTP request, irrespective of requiring PHP, the new httpd will require 5.8 Megabytes of memory to be allocated to service that client request instead of allocating 1.1 Megabytes of memory. If there were 100 simultaneous HTTP requests, the server would have to allocate over a half of gigabyte of memory to service the requests.

The DSO PHP implementation used in the Red Hat 8 environment is not possible until the MYSEA httpd is ported to provide the correct output required by the Apache ‘apxs’ support program. However, this method uses the same amount of memory as the static method; therefore, this implementation is of no real advantage with respect to memory usage. Another alternative was to compile PHP as a Common Gateway Interface (CGI) binary. When used as a CGI binary, the PHP interpreter is used as a standalone program, which is only invoked when a HTTP request involves a PHP web page. The httpd binary remains unchanged, only the configuration file (http.conf) is altered to handle the servicing of PHP web pages. This method uses fewer memory resources because the PHP interpreter is only invoked when a HTTP request involves parsing PHP web pages.

There are two common methodologies employed when using PHP as a CGI binary. If the PHP binary is to be placed outside of the web directory structure, for example placing PHP in /usr/local/bin, then every PHP web page will require the directive ‘#!/usr/local/bin/php’ to be placed in the first line of the PHP script. The other method is to place the PHP binary in the web directory structure in the cgi-bin directory; however, in this case all PHP pages will need to be placed under this directory as well.

Using PHP as a CGI binary requires less memory since the PHP interpreter is only invoked when a HTTP request involves parsing a PHP web page, however, there is an overhead in processing and response time since the PHP interpreter is called for every PHP web page request. Properly securing

PHP to function as a CGI binary is another issue. CERT Advisory CA-1996-11 [30] states that no interpreters should be placed in the cgi-bin directory. If an attacker gained access to the cgi-bin directory, he could remotely execute any command that the interpreters can execute on the server. Due to the unique nature of the MYSEA architecture's MLS security configuration, further analysis is required to determine whether this would be a security concern.

A possible security risk exists due to the method of placing the PHP binary outside of the web directory structure. If a PHP web page was requested by any user and the PHP interpreter failed to parse that requested web page the output would be the source code of that PHP web page rendered as clear text. This could potentially lead to a loss of proprietary or sensitive information.

The CGI method was not pursued due to time constraints in testing the security of the CGI implementation in conjunction with the PHP code that comprises the BASE web application.

E. FUNCTIONAL TEST PLAN AND REPORTS

This section will cover a comprehensive functional test plan that will demonstrate the functionality of the IDS components in both environments. The installation instructions in Appendix A and Appendix B include component functional testing; therefore, that level of testing will not be addressed in this test plan. The functional test plan described in this section uses a test program called IDSWakeup [31] to provide a variety of alerts and to establish a baseline to determine how many alerts each sensor should detect. Both the Red Hat 8 Linux and XTS400 environments were tested. Appendix C contains the detailed steps for conducting each test. Figure 12 shows the physical layout of the machines used for these tests.

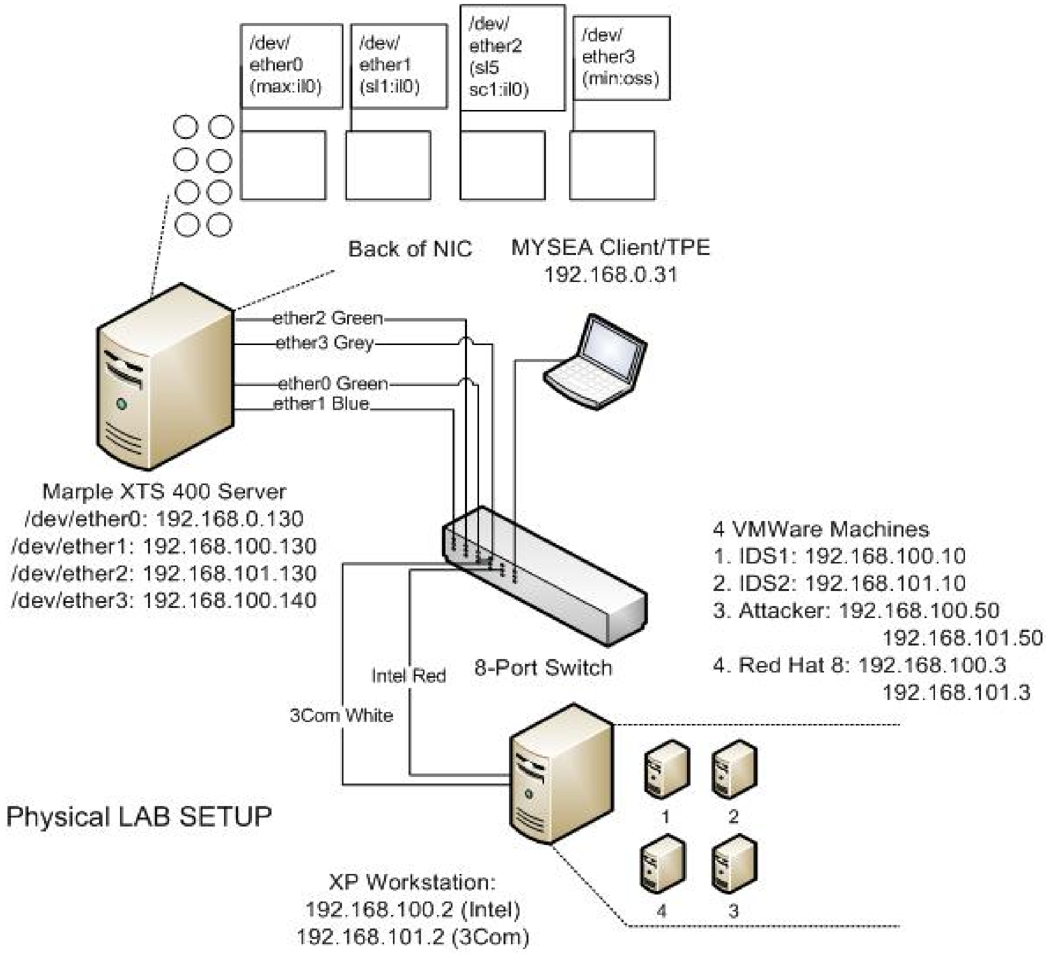


Figure 12: Lab Setup

1. Test Plan Objectives: Red Hat 8 Linux and XTS400 Environments

The first objective of the functional testing is to verify that the IDS sensor, Snort, is actually monitoring the traffic and looking for malicious traffic. When malicious traffic is discovered, Snort should send an alert via the PostgreSQL client resident on the sensor to the remote PostgreSQL database. The second objective is to verify that new alerts are stored in the database, and that the data is accessible by the BASE web application for displaying information pertaining to the alerts. A third objective is to ensure the segregation of network traffic between the sensors and their respective databases. Traffic monitored by IDS1 should not be seen by IDS2 and vice versa. To accomplish these testing objectives the test program IDSWakeup will be used to generate fake attacks.

IDSWakeup is a false positive generator that sends IP packets containing fake malicious data to a destination IP address. Snort construes this data as actual malicious data, thus causing an alert to trigger. Not all of the fake alerts generated by IDSWakeup may be suitable for testing Snort. The false attacks are stateless attacks, which mean there is no established TCP/IP connection between the remote host and the machine sending the false attacks. The current version of Snort is better at determining if these stateless attacks warrant an alert over the previous versions of Snort. To illustrate this point, R. Bejtlich [34], used IDSWakeup to test Snort 2.6.1.5. Out of 181 packets sent by IDSWakeup, only 26 alerts were generated by Snort.

The method for the test was to attack a system on the same network segment as the sensors IDS1 and IDS2, thus testing that IDS1 and IDS2 are properly monitoring the network traffic. To conduct the test another Debian machine was created (hostname Attacker) to serve as the attack platform and configured with the IDSWakeup program.

2. Functional Test: Red Hat 8 Linux Environment

Before commencing the test, each component was verified to be functioning properly. Refer to the installation instructions in Appendix A or Appendix B for performing component-level functional tests.

The first test sent the fake attacks to the unclassified IP address of the Red Hat 8 server (192.168.100.3). The expected result is that IDS1 should detect some attacks and send alerts to its respective PostgreSQL database on the Red Hat 8 server. The number of attacks sent by IDSWakeup that IDS1 will actually identify as attacks was unknown so the first attempt served as a baseline. IDS2 should not send alerts because it should not see any of the fake attacks if the networks are properly separated.

The attack machine was reconfigured for the secret network and the test was repeated, sending the fake attacks to the secret IP address on the Red Hat 8 server (192.168.101.3). The expected result for this test was that IDS2 should detect the same number and type of attacks that IDS1 detected and the alert count on IDS1 should not increase.

3. Functional Test Report: Red Hat 8 Linux Environment

After IDSWakeup completed its first test run against IDS1, the unclassified web site BASE was accessed. Table 4 shows the breakdown of alerts detected by Snort as viewed on the home page of the BASE unclassified website. The secret BASE web site showed zero alerts, which is the expected result and verifies that network traffic was properly segmented.

At the completion of the second test run, the secret BASE web site showed that the same type and number of alerts detected by IDS1 were detected by IDS2. Table 5 shows the data recorded from this test. Accessing the unclassified BASE web site showed that the previous alert count did not increase, therefore the network traffic was properly segmented and each IDS sent alert data to the proper PostgreSQL database.

RH8 IDS1

Total # Alerts	Unique Alerts	TCP%	UPD%	ICMP%	Portscan Traffic %
18	12	56%	22%	11%	11%

Table 4: IDS1 Alerts Sent

RH8 IDS2

Total # Alerts	Unique Alerts	TCP%	UPD%	ICMP%	Portscan Traffic %
18	12	56%	22%	11%	11%

Table 5: IDS2 Alerts Sent

Using the test program IDSWakeup the following test objectives were verified:

a. IDS sensor can monitor traffic in promiscuous mode

b. IDS sensor can detect known malicious traffic and generates valid alerts

c. Network traffic is properly segmented

d. Alerts are properly stored in the proper database

e. Alert data is accessible from the BASE application

4. Functional Test: XTS400 Environment

The same testing methodology was repeated for the XTS400 environment using the same IDS sensors that were used in the Red Hat 8 Linux environment. To counteract denial of service attacks Snort uses threshold times for sending alerts. If the IDSWakeup test were looped, Snort would eventually stop sending alerts. This factor was taken into account when these tests were repeated soon after testing the Red Hat 8 environment, however, due to the time required to reconfigure the Attacker machine and low frequency of test runs, there was no expectation that the threshold times would be reached. The expected results are that IDS1 and IDS2 will detect the same number and type of attacks but alerts will be sent to the databases on the XTS400 system.

The enforcement of MAC policies will also be tested by using the user ‘mdemo1’ that was configured during the MYSEA environment installation to access the unclassified BASE web site and secret BASE web site. The standard MYSEA installation procedures were used to set up the test environment. Since the Attacker machine was still configured for the secret network, the testing started by running the IDSWakeup test against IDS2. To view alerts a SIM_SECRET session was established for user mdemo1 and the secret BASE web site was accessed to view the alerts.

The Attacker machine was reconfigured to operate on the unclassified network and the IDSWakeup test was repeated, this time against IDS1. To access the unclassified web site a SIM_UNCLASSIFIED session was established for the user mdemo1.

5. Functional Test Report: XTS400 Environment

The data recorded in Tables 6 and 7, obtained by viewing each BASE web application, shows that the sensors were able to detect malicious traffic and send alerts about that malicious traffic to their respective databases.

To test MAC poli cies an attempt to access the secret BASE web site was made while the user mdemo1 was logged in under a SIM_UNCLASSIFIED session. The result was a web page informing that access to the BASE web site was forbidden.

XTS400 IDS1

Total # Alerts	Unique Alerts	TCP%	UPD%	ICMP%	Portscan Traffic %
16	10	63	25	13	0

Table 6: IDS1 Alerts Sent

XTS400 IDS2

Total # Alerts	Unique Alerts	TCP%	UPD%	ICMP%	Portscan Traffic %
16	10	63	25	13	0

Table 7: IDS2 Alerts Sent

Using the IDSWakeup program and attempting to violate MAC policies verified the following objectives:

a. IDS sensor can monitor traffic in promiscuous mode

b. IDS sensor detected known malicious traffic and generates valid alerts

c. Network traffic is properly segmented

d. Alerts are properly stored in the proper database

e. Alert data is accessible from the BASE application

f. MAC policies are enforced by the MYSEA programs and STOP 6.3 OS with respect to database accesses and web site access

F. SUMMARY

Implementing the PostgreSQL databases and BASE web application on the MLS server required some customization because of the MAC policies employed on the STOP 6.3 operating system and previous modifications to the Apache source code. The implementation of the IDS components was successful and the functional test verified that the IDS components functioned as expected in both environments. The next chapter presents a conclusion about the overall contribution of IDSes in the MYSEA architecture and possible future work.

THIS PAGE INTENTIONALLY LEFT BLANK

VI CONCLUSIONS AND FUTURE WORK

This chapter states the conclusions of this thesis and discusses possible future work related to IDS integration in the MYSEA architecture.

A. CONCLUSION

The goal of this thesis was to provide a method to integrate an Intrusion Detection System into the MYSEA architecture and provide security analysts with an analysis tool, accessible from any workstation within the MLS network, to analyze all alerts sent by the IDS sensors. Results from background research on IDSes established that the typical deployment of IDSes uses the three components that were implemented in this project: sensors, databases, and analysis tools. Using a formal selection process, the sensor selected for this project was the open source product Snort from Sourcefire. To store alerts sent from Snort, the open source database product PostgreSQL was chosen. To view alert data in a tractable format, the open source product Basic Analysis Security Engine (BASE) was used.

Implementation of these components in the MYSEA architecture constituted the main challenge of this work. Several designs were considered. The first design, based on preliminary testing, utilized one central database, however due to complications associated with trying to move data between networks at different classification levels, an alternative design was created that used multiple databases, one for each sensor. This design allows use of the tool BASE to analyze network traffic and formulate a complete picture concerning probable network attacks.

The preferred design, which supported storage of all alerts in one external central database, was altered due to complications with the installation of a key component, the MySQL proxy software, on the STOP 6.3 operating system. Instead of one central repository each IDS sends data to its own remote database, which is installed on the MLS server. The project clearly demonstrated that it was possible to implement the required components to create a functional Intrusion Detection System.

The current implementation may not be the optimal implementation, but the MYSEA architecture is constantly undergoing development and refinement. It is to be expected that as the MYSEA architecture moves from STOP operating system version 6.3 to version 7.0 there will be significant improvements to the IDS implementation method.

B. FUTURE WORK

The following issues became known during this project and warrant further study.

1. Labeling Data

In the Concept of Operations section, an assumption was made that the database is benign and does not alter the security classification labels that were applied by the ML process. The ML process (see Figure 5) receives data from processes U and S and applies a label to the data at the application layer with a security classification before the data is forwarded to process Sys High. The protection of the classification labels from malicious or unintended permutations should be addressed.

The ML process could provide an integrity check through the use of "crypto seals" applied to the embedded labels. As data is transferred between the database and MYSEA client machine used by the analyst, the ML process would verify the integrity of the data by checking the crypto seal.

2. MySQL Proxy Program

The MySQL Proxy code used in preliminary testing is neoteric code. The problem with compiling and implementing the MySQL Proxy was covered in Chapter IV, Section D. It was determined through testing on a Red Hat 8 system that this code would be incompatible with the existing libraries on the STOP 6.3 operating system.

The STOP 7 operating system may support the programs and libraries required to compile the MySQL proxy code.. Therefore, research could be applied to the feasibility in implementing the MySQL proxy code on the STOP 7 operating system. This proxy code may yield some benefits to future IDS implementations.

3. Read Down Support

During installation and testing of the BASE web sites, an observation was made that when a session is established at SIM_SECRET the user of that session cannot access data from the unclassified Snort database via the unclassified BASE application. At the SIM_SECRET session level, the user should be able to perform a “read down” of the data in the unclassified Snort database. When an attempt was made to access data from the unclassified Snort database via the unclassified BASE application, the result was a database connection error. Full details of this observation are covered in Chapter V, Section D.

For the analysis tool, BASE, to exhibit read down capabilities, modifications would have to be made to the code. The current application makes SQL queries to the database, which require both, read and write access. Research is required to either modify the database and BASE application to support read down capabilities or to implement a trusted database proxy.

4. PHP Implementation

Neither implementation of PHP, either as a DSO module, nor as a static module, is an optimal solution for the MYSEA environment due to the amount of memory each implementation uses. The security issues related to using PHP as a CGI binary resulted in a suboptimal solution as well.

One area for future research is the use of light and heavy httpd binaries. When dynamic content such as PHP is required, the light httpd could proxy to the heavy httpd to handle the request. The heavy httpd would be compiled with all required modules such as mod_perl, mod_php, and so on. The light httpd is only compiled with the minimum modules required to create a functional httpd binary. One required module would be the proxy module. This method would eliminate the security risks of using an interpreter such as CGI, and potentially would save memory resources. Only the processing delay between the light httpd and heavy httpd is a potential issue.

APPENDIX A RED HAT 8 LAB INSTALLATION

This appendix outlines the installation procedures for installing all of the IDS components using the Linux Debian 4.0 operating system and Linux Red Hat 8 operating system. Apache, PostgreSQL, and Basic Analysis Security Engine (BASE) will be installed on the Red Hat 8 and Snort will be installed on Debian 4.0.

To view the BASE web site a client with a web browser such as Internet Explorer or Firefox is also required. .

A. INSTALLATION AND TEST TOPOLOGY

The test topology is comprised of three physical machines and four virtual machines (VMs) built using VMWare Workstation from VMWare. Figure 13 shows the topology that was used to perform the installation and tests for this project.

Dell workstation:

2.8 GHz CPU with 2GBs RAM

Two physical network ports

Four virtual machines with virtual bridging

Bridge 0: IDS1, Red Hat 8, Attacker

Bridge 1: IDS2, Red Hat 8, Attacker

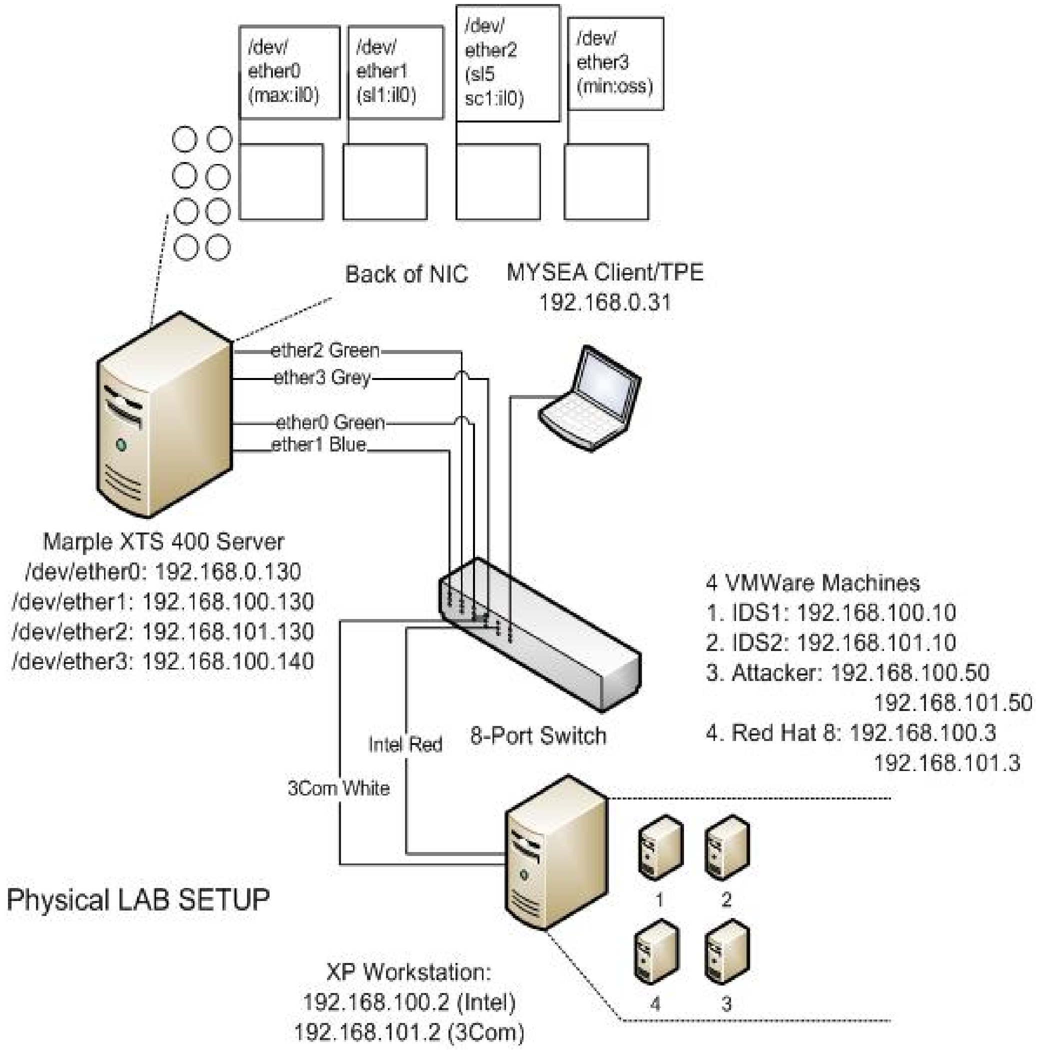


Figure 13: Lab Setup

Installing the desktop operating system, Red Hat 8, VMWare, and configuring VMWare workstation is beyond the scope of this project.

1. Conventions used in this Documentation

All commands that need to be typed at the command prompt are *italicized* and in *`Courier New`* font. All items that are typed into a configuration file are in `Courier New` font.

B. INSTALLING SNORT ON DEBIAN 4.0

This section covers installing the Debian, Snort, and other programs required to support Snort.

1. Debian Operating System Installation

1. Insert Debian DVD-1 and boot up the PC or VM

2. Select english for language

3. Select United States for country

4. Select American English for keymap

5. DO NOT USE DHCP, configure the network manually. Enter the following information (refer to lab diagram for IP addresses)

IP: IDS1 - `192.168.100.10` IP: IDS2 - `192.168.101.10`

Net Mask for both IDS1 and IDS2: `255.255.255.0`

Gateway <leave blank>

DNS server <leave blank>

6. Enter a hostname: For IDS1: `IDS1` For IDS2: `IDS2`

7. Enter domain name: <leave blank>

8. Partition Disk:

Select Guided - use entire disk

Partitioning scheme - select Separate /home, /usr, /var, and /tmp partitions

Install will compute partition space. Select Finish partitioning and write changes to disk then yes to commit changes

9. Configure time zone, set for PST

10. Set root's password

11. Add a user and password

12. Install will set up base system

13. Select no for network mirror, all software should be installed from one of the three DVDs or from CD1

14. Select no to participate in package survey

15. Software selection

Unselect Desktop environment

Unselect Standard system.

16. Select Yes to install GRUB boot loader to the master boot record

17. Reboot into the finished system

2. Support Software Installation

Use `apt-get install` to install software from the Debian DVD.

Example lets install ssh so a ssh terminal can be used if needed.

1.
```
apt-get install ssh
```

Follow the prompts to complete the installation

2. Install the following packages from DVD-1. These will be required for installing libpcap and snort

gcc g++ make postgresql-client flex bison libpcre3 libpcre3-dev

libpq-dev

```
apt-get install gcc g++ make postgresql-client flex bison libpcre3 libpcre3-dev libpq-dev
```

3. Libpcap0.9.8 Installation

1. Use libpcap0.9.8.tar.gz file provided on CD1 /snort-2.8.0.1

2. Copy tar file to /usr/local/src

```
 cd /usr/local/src
tar –xvzf libpcap-0.9.8.tar.gz
 cd libpcap-0.9.8
 ./configure
 make
 make install
```

4. Snort Installation

1. Use the snort tar file provided on CD1 /snort-2.8.0.1

2. Copy tar file to /usr/local/src

```
 cd /usr/local/src
 tar –xvf snort-2.8.0.1.tar
 cd snort-2.8.0.1
./configure --with-postgresql --enable-dynamicplugin
 make
 make install
 mkdir /etc/snort
 mkdir /var/log/snort
 groupadd snort
 useradd –g snort snort
 chown snort:snort /var/log/snort
```

5. Snort Rules Installation

1. Use the snort rules tar file provided on CD1 /snort-2.8.0.1

2. Copy the rules tarball to /etc/snort

```
 cd /etc/snort
 tar –xvzf snortrules-snapshot-CURRENT.tar.gz
 cp /usr/local/src/snort-2.8.0.1/etc/*.conf* /etc/snort
 cp /usr/local/src/snort-2.8.0.1/etc/*.map  /etc/snort
```

3. Configure the snort.conf file located at /etc/snort/snort.conf. Change the following lines less the quotes:

`‘var HOME_NET any’` change to:

```
‘var HOME_NET 192.168.100.0/24’     #for IDS1
‘var HOME_NET 192.168.101.0/24’     #for IDS2
```

`‘var EXTERNAL_NET any’` change to:

```
‘var EXTERNAL_NET !$HOME_NET’
```

Search on the string ‘var RULE_PATH’, change the line to:

```
‘var RULE_PATH /etc/snort/rules’
```

Functional Test

1. Start snort with the following command

```
snort –D –u snort –g snort –c /etc/snort/snort.conf
```

2. Check that eth0 enters promiscuous mode

3. Check /var/log/syslog for Snort initialization completed successfully (pid=some number)

```
tail -f /var/log/syslog
```

4. Run a portscan on the IDS using nmap or any available port scanner.

check /var/log/snort/alerts for a portscan priority 3 message

```
tail -f /var/log/snort/alerts
```

5. If snort works then kill the snort process and continue

```
    ps aux | grep snort
    kill <pid number of snort process>
```

End Functional Test

6. Finalize Sensor Installation

Change configuration file and send alerts to Red Hat

STOP!

NOTE: Before proceeding with these steps, complete the installation procedures for PostgreSQL including setting up the schema for the snort database.

Functional Test

1. Run the following command:

```
psql -h 192.168.100.3  -U snort -d snort
```

NOTE: The IP address will vary according to your network configuration

2. **<Observe**: should get a snort=> prompt. Exit from psql with \q

End Functional Test

3. Edit snort.conf to send all alerts to the PostgreSQL database

```
cd /etc/snort
```

edit snort.conf

page down to Step #4: Configure output plugins

find the line # output database: alert, postgresql…

NOTE: search for postgres with your editor's search feature

Configure the parameters so the line looks like this:

```
output database: log, postgresql, user=snort
dbname=snort port=5433 host=192.168.100.3
```

NOTE: your parameters will vary based on your network settings and port settings:

RH8 network 1: 192.168.100.3

RH8 network 2: 192.168.101.3

Unclassified Postgresql database: port 5433

Secret Postgresql database: port 5434

Functional Test

4. Start snort and check that it initialized correctly:

```
snort –D –u snort –g snort –c /etc/snort/snort.conf
tail –f /var/log/syslog
```

5. Repeat the portscan test. This time the alert should go to the PostgreSQL database. Log into the PostgreSQL database either remotely from the IDS or locally on the database server:

```
snort=> select * from event;
snort=> select * from signature;
```

Depending on the port scanner used there may be one or two events and one or two signatures.

End Functional Test

7. Lokkit Firewall Installation

To give the sensor some protection install a simple iptables based firewall called lokkit

1. Copy lokkit_0.50.22-7.1_i386.deb from CD1 /misc to /usr/local/src

```
dpkg –i lokkit lokkit_0.50.22-7.1_i386.deb
```

2. Run the command *lokkit*
3. Choose medium and customize

Allow port 22 for SSH and the port for the PostgreSQL database you wish to send alerts. The database port must be added for two-way communication between database server and IDS.

C. INSTALLING POSTGRESQL 7.4.18 ON RED HAT 8

1. Create System Accounts

Create two users; **postgres** and **snort.** User **postgres** will own the postmaster process and **snort** is used to make the databases. Neither user requires any elevated privileges.

```
adduser postgres
adduser snort
```

change password for each user

passwd <username> then follow the prompts

2. PostgreSQL 7.4.18 Installation

Perform the following steps as **root**:

1. Copy postgresql-base-7.4.18.tar.gz to /usr/local/src from CD1 /postgres-7.4.18

```
tar -xvzf postgresql-base-7.4.18.tar.gz
cd postgresql-7.4.18
```

NOTE: Compiling Postgresql requires gmake > 3.76.1

gmake --version = 3.79.1 on RH

ISO/ANSI C compiler such as GCC

GNU Flex and Bison

2. Run configure

```
./configure
```

Default installation is installed at /usr/local/pgsql. All other files will install to sub-directories under /pgsql

Use *./configure --help* to list other alternatives

3. Start build with gmake

```
gmake
```

<Observe: "All of PostgreSQL successfully made. Ready to install"

4. (optional) Regression Tests

As a non-privileged user:

```
gmake check
```

5. Install the files

```
gmake install
```

<Observe: Installed with no errors>

3. Post-installation Setup

1. Update Environment Variables

add the following two lines to /etc/profile

```
PATH=/usr/local/pgsql/bin:$PATH
MANPATH=/usr/local/pgsql/man:$MANPATH
```

2. Initialize a database storage area on disk

```
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su postgres
initdb -D /usr/local/pgsql/data
```

<observe:

The files belonging to this database system will be owned by user "postgres"…

>

3. Starting and stopping the database server as user postgres

Use the pg_ctl script as user postgres

Type *pg_ctl --help* for options

Start:

```
pg_ctl start -D  /usr/local/pgsql/data -l serverlog
```

Stop:

```
pg_ctl stop -D /usr/local/pgsql/data -m fast
```

Functional Test

4. Start the database and test the setup by making a database called mydb then login as postgres. PostgreSQL 7.4.18 requires logging in with username and database unless username and database are the same name.

```
pg_ctl start -D /usr/local/pgsql/data -l logfile
createdb mydb
psql -U postgres -d mydb
\q
```

Stop the database

```
pg_ctl stop -D /usr/local/pgsql/data -m fast
exit
```

(to exit the su postgres shell)

End Functional Test

5. Setting up to simulate multi-level access on a STOP 6.3.1 OS:

Make the following directories as user root:

```
mkdir /var/postgresql
mkdir /var/postgresql/unclass_data
mkdir /var/postgresql/secret_data
```

Change owner to user postgres:

```
chown postgres /var/postgresql/unclass_data
chown postgres /var/postgresql/secret_data
su postgres
initdb -D /var/postgresql/unclass_data
initdb -D /var/postgresql/secret_data
```

Edit each postgresql.conf file

On both files enable tcpip connections: `tcpip_socket = true`

Give each instance a different port number

unclass_data: `port = 5433`

secret_data: `port = 5434`

Edit pg_hba.conf for each data cluster area. Page down to the end of the file.

/var/postgresql/unclass_data/pg_hba.conf

```
TYPE: host
DATABASE: snort
USER: snort
IP_ADDRESS: 192.168.100.0
MASK: 255.255.255.0
AUTH_TYPE: MD5
```

/var/postgresql/secret_data/pg_hba.conf

```
TYPE: host
DATABASE: snort
USER: snort
IP_ADDRESS: 192.168.101.0
MASK: 255.255.255.0
AUTH_TYPE: MD5
```

Functional Test

Start the unclass data area from directory /var/postgresql/unclass_data

```
cd /var/postgresql/unclass_data
pg_ctl start -D /var/postgresql/unclass_data -l logfile
```

Check the logfile, ensure database is ready

Start the secret data area from directory /var/postgresql/secret-data

```
cd /var/postgresql/secret_data
pg_ctl start -D /var/postgresql/secret_data -l logfile
```

Check the logfile, ensure the database is ready

End Functional Test

6. Create the snort databases for each sensor

As the user postgres create the user snort for each data area

```
createuser -p 5433 snort
```

Select yes to create databases and no to create other users

```
createuser -p 5434 snort
```

Select yes to create databases and no to create other users

Alter snort’s database password for unclass_data

```
psql -p 5433 -U postgres -d template1
alter user snort password ‘mysea’;
\q
```

Alter snort’s database password for secret_data

```
psql -p 5434 -U postgres -d template1
alter user snort password ‘mysea’;
\q
```

Create the snort databases

```
su - snort
 createdb -p 5433 snort
 createdb -p 5434 snort
```

Update the schema for each database. Obtain the create_postgresql file from CD1 /snort-2.8.0.1/create_postgresql and copy to a temporary directory.

Change to the directory containing ‘create_postgresql’ file

```
psql -p 5433 < ./create_postgresql
psql -p 5434 < ./create_postgresql
```

Functional Test

7. If the sensors have been completed:

Login from sensor IDS1

```
psql -h 192.168.100.3 -p 5433 -U snort -d snort
```

Enter password

Run a query

```
select * from event;
```

Should get 0 rows.

\q

NOTE: If the output is, access denied the database is not owned by snort

Login from sensor IDS2

```
psql -h 192.168.101.3 -p 5434 -U snort -d snort
```

Enter password

Run a query

```
select * from event;
```

Should get 0 rows.

\q

NOTE: If the output is, access denied the database is not owned by the user snort

Exit from all of the shells to get back to the root user but leave both databases running.

End Functional Test

D. INSTALLING BASE ON RED HAT 8

This section covers installing the web application BASE, along with required software programs Apache and PHP. Perform the following procedures as **root**

1. Apache Installation

1. Install Apache 1.3.0 from CD1. Copy apache_1.3.0.tar.tar from CD 1 /apache-1.3.0 to /usr/local/src

```
cd /usr/local/src
tar -xvzf apache_1.3.0.tar.tar
cd /apache_1.3.0
```

NOTE: To use PHP with Apache, Apache needs to be compiled with mod_so support.

```
./configure --enable-module=so
make
make install
```

<observe: the banner "You now have successfully built and installed the Apache 1.3..." box>

2. Edit httpd.conf

Edit httpd.conf located at /usr/local/apache/etc

Change ServerName to be `localhost` or IP address of the machine

3. Starting and Stopping apache server

start apache:*/usr/local/apache/sbin/apachectl start*

stop apache: */usr/local/apache/sbin/apachectl stop*

Functional Test

4. Test the server

Start the apache server:

```
/usr/local/apache/sbin/apachectl start
```

Using a web browser that can access the RH8 server browse to http://192.168.100.3 or whatever IP address is used by RH8.

<Observe: The "It Worked!" home page should be displayed>

5. Verify requirements for installing php4

```
/usr/local/apache/sbin/httpd -l
```

Look for two modules: http_core.c and mod_so.c

If they are present proceed to INSTALL PHP4 otherwise troubleshoot why the modules were not compiled.

2. PHP4.3.11 Installation

1. Install PHP 4.3.11 from CD1

OPTIONAL: If you want graphic support for BASE, i.e., graphs and charts, perform the following installation procedures:

Copy all of the files from CD1 /php-4.3.11/GD support to /usr/local/src and install them using the rpm package installer.

```
rpm -U libpng-1.2.2-8.i386.rpm
rpm -iv libpng-devel-1.2.2-8.i386.rpm
rpm -iv libjpeg-devel-6b-21.i386.rpm
rpm -iv zlib-1.1.4-4.i386.rpm
rpm -iv zlib-devel-1.1.4-4.i386.rpm
```

Copy php-4.3.11.tar.tar from CD1 /php-4.3.11 to /usr/local/src

```
tar -xvzf php-4.3.11.tar.tar
cd php4.3.11
./configure --with-apxs=/usr/local/apache/sbin/apxs --with-pear --with-pgsql=/usr/local/pgsql --with-gd --with-png-dir=/usr/lib --with-jpeg-dir=/usr/lib --with-zlib-dir=/usr/include
```

NOTE: this is with graphics support. If you do not want graphics support (GD) them omit everything after --with-gd

```
make
make install
```

NOTE: ignore the suggestion about adding the path to the php.ini file

2. Post Installation

Edit /usr/local/apache/etc/httpd.conf

Add the following to the end of the file

```
#AddType directives for php 4.3.1
AddType application/x-httpd-php .php .phtml
AddType application/x-httpd-php-source .phps
```

Restart apache:

```
/usr/local/apache/sbin/apachectl restart
```

Functional Test

Create a test script called phpinfo.php

```
          <? phpinfo() ?>
```

Save phpinfo.php and place in apache's document root

```
cp phpinfo.php /usr/local/apache/share/htdocs
```

Browse to http://192.168.100.3/phpinfo.php

<observe: a web page that displays php information>

End Functional Test

Optional install for making Charts and Graphs in BASE

If PHP was compiled with GD support perform the following:

1. Copy all files from CD1 /php-4.3.11/pear to /usr/local/src

```
cd /usr/local/src
```

2. Run the command: *pear config-set preferred_state alpha*

3. Install the following packages in order

```
pear install Image_Color-1.0.2.tar
pear install Image_Canvas-0.3.0.tar
pear install Image_Graph-0.7.2.tar
pear install Numbers_Roman-1.0.1.tar
pear install Numbers_Words-0.14.0.tar
```

3. BASE (Basic Analysis Security Engine) Installation

1. Install ADOdb. Copy adodb498.gz from CD1/base-1.4.0 to /usr/local/

```
tar -xvzf adodb498.gz
```

<observe: new directory adodb created with files>

NOTE: There is nothing to install but remember this location for configuring BASE

4. Unclassified BASE Installation

NOTE: verify the unclass database snort is running before performing these procedures

1. Copy base 1.4.0.tar.gz from CD1 to /usr/local/apache/share/htdocs

```
cd /usr/local/apache/share/htdocs
tar -xvzf base-1.4.0.tar.gz
2. Rename directory base-1.4.0 to base_unclass:
mv base-1.4.0 base_unclass
3. Set read/write/execute on directory base_unclass
chmod 777 base_unclass
```

2. Open a web browser and browse

to http://192.168.100.3/base_unclass/setup/indiex.php

<Observe: web page with banner and Settings:

Configurable writing: yes

Php version: 4.3.11

Php logging level: <blank>

5. Step 1 of 5

Enter the location to adodb: `/usr/local/adodb`

6. Step 2 of 5

Enter the applicable information

Pick a Database type: `PostGRES`
Database name: `snort`
Database host: `192.168.100.3`
Database port: `5433`
Database user name: `snort`
Database password: `mysea`
Leave the Archive Database items blank

Click Submit Query

7. Step 3 of 5

Click the box next to Use Authentication System

Admin user name: `snort`
Password: `mysea`
Full name: `snort`
NOTE: Any name can be used

Click on Submit Query

8. Step 4 of 5

Click on Create BASE AG button to extend the snort database

Verify all tables were successfully created and roles inserted

Click on the link Now continue to step 5 ... (middle of the page)

9. Login as `snort` password `mysea`.

<Ignore the error about user not existing>

10. Reset the attributes on the base_unclass directory

```
chmod 755 base_unclass
```

11. Cosmetic change: Change style sheets so all banners display the word “UNCLASS” at the top of each page.

```
cd /usr/local/apache/share/htdocs/base_unclass/languages
```

Edit english.lang.php

Search for _TITLE

Add the word UNCLASS before and after the title in single quotes

12. Bug fix: Fixes fatal error when log out link is clicked.

```
cd /usr/local/apache/share/htdocs/base_unclass
```

Open the base_logout.php file in a text editor

Change ‘`base_header()`’ function to ‘`header()`’

Keep all text inside the parentheses

Save the file

<u>NOTE:</u> If you did the IDS testing then you should see some alert data in the database.

5. Secret BASE Installation

<u>NOTE:</u> verify the secret database snort is running before performing these procedures

1. Return to the htdocs directory and untar base

```
cd /usr/local/apache/share/htdocs
tar -xvzf base-1.4.0.tar.gz
```

2. Rename directory base-1.4.0 to base_secret:

```
mv base-1.4.0 base_secret
```

3. Change read/write/execute attributes on directory base_secret

```
chmod 777 base_secret
```

4. Open a web browser and browse

to: http://192.168.101.3/base_secret/index.php

<Observe: web page with banner and Settings:

Configurable writing: yes

Php version: 4.3.11

Php logging level: <blank>

5. Step 1 of 5

Enter the location to adodb: `/usr/local/adodb`

6. Step 2 of 5

Enter the applicable information

Pick a Database type: `PostGRES`
Database name: `snort`
Database host: `192.168.101.3`
Database port: `5434`
Database user name: `snort`
Database password: `mysea`

Leave the Archive Database items blank

Click Submit Query

7. Step 3 of 5

Click the box next to Use Authentication System

Admin user name: `snort`
password: `mysea`
full name: `snort`
NOTE: Any name can be used

Click on Submit Query

8. Step 4 of 5

Click on Create BASE AG button to extend the snort database

Verify all tables were successfully created and roles inserted

Click on the link Now continue to step 5 ... (middle of the page)

9. Login as `snort` password `mysea`.

<Ignore the error about user not existing>

10. Reset the attributes on directory base_secret

```
 chmod 755 base_secret
```

11. Cosmetic change: Change style sheets so all banners display the word "SECRET" at the top of each page.

```
cd /usr/local/apache/share/htdocs/base_secret/languages
```

Edit english.lang.php

Search for _TITLE

Add the word SECRET before and after the title in single quotes

12. Bug fix: Fixes fatal error when log out link is clicked.

```
cd /usr/local/apache/share/htdocs/base_secret
```

Open the base_logout.php file in a text editor

Change 'base_header()' function to 'header()'

Keep all text inside the parentheses

Save the file

NOTE: If you did the IDS testing then you should see some alert data in the database.

APPENDIX B XTS400 INSTALLATION PROCEDURES

This appendix outlines the installation procedures for the IDS on a Debian 4.0 operating system and installing PostgreSQL and Basic Analysis Security Engine (BASE) on the STOP 6.3 operating system.

A. INSTALLATION AND TEST TOPOLOGY

The test topology is comprised of three physical machines and two Virtual Machines. One of the physical machines is the XTS400 server that runs the STOP 6.3 operating system. The diagram above the server shows the layout of the 4-port network interface card mounted in the PCI bus at the rear of the machine. This is important to information to keep in mind when setting up the test network and configuring the Ethernet cards. Figure 14 shows the lab layout.

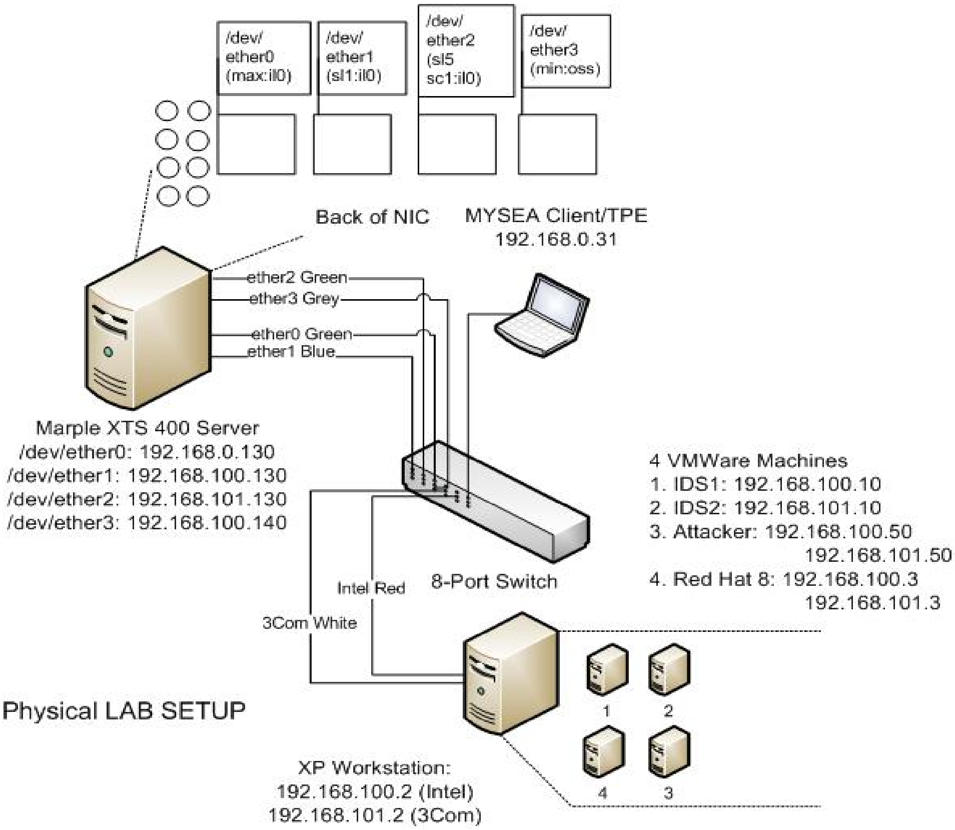


Figure 14: Lab Setup

Installing the desktop operating system, Red Hat 8, VMWare, and configuring VMWare workstation is beyond the scope of this project.

1. Conventions used in this Documentation

All commands that need to be typed at the command prompt are *italicized* and in `Courier New` font. All items are typed into a configuration file are in `Courier New` font.

B. INSTALLING SNORT ON DEBIAN 4.0

This section covers installing the Linux Operating System, Snort, and other programs required to support Snort.

1. Debian Operating System Installation

1. Insert Debian DVD-1 and boot up the PC

2. Select english for language

3. Select United States for country

4. Select American English for keymap

5. DO NOT USE DHCP, configure the network manually

Enter the following information (refer to lab diagram for IP addresses)

IP: IDS1 - `192.168.100.10` IDS2 - `192.168.101.10`
Net Mask: For both IDS - `255.255.255.0`
Gateway: <leave blank>
DNS server: <leave blank>

6. Enter a hostname: For IDS1 - `IDS1` For IDS2 - `IDS2`

7. Enter domain name: <leave blank>

8. Partition Disk

Select Guided - use entire disk

Partitioning scheme - select Separate /home, /usr, /var, and /tmp partitions

Install will compute partition space. Select Finish partitioning and write changes to disk then yes to commit changes

9. Configure time zone, set for PST

10. Set root's password

11. Add a user and password

12. Install will set up base system

13. Select no for network mirror, all software should be installed from one of the three DVDs or from CD1

14. Select no to participate in package survey

15. Software selection

Unselect Desktop environment

Unselect Standard system.

16. Select Yes to install GRUB boot loader to the master boot record

17. Reboot into the finished system

2. Support Software Installation

Use *`apt-get install`* to install software.

Example lets install ssh so a ssh terminal can be used if needed.

1. *`apt-get install ssh`*

2. Install the following packages from DVD-1. These will be required for installing libpcap and snort.

gcc g++ make postgresql-client flex bison libpcre3 libpcre3-dev

libpq-dev

```
    apt-get install gcc g++ make postgresql-client flex
bison libpcre3 libpcre3-dev libpq-dev
```

3. Libpcap0.9.8 Installation

1. Use libpcap0.9.8.tar.gz file provided on CD1 /snort-2.8.0.1

2. Copy tar file to /usr/local/src

```
cd /usr/local/src
 tar –xvzf libpcap-0.9.8.tar.gz
 cd libpcap-0.9.8
 ./configure
 make
 make install
```

4. Snort Installation

1. Use the snort tar file provided on CD1 /snort-2.8.0.1

2. Copy tar file to /usr/local/src

```
cd /usr/local/src
 tar –xvf snort-2.8.0.1.tar
 cd snort-2.8.0.1
./configure --with-postgresql --enable-dynamicplugin
 make
 make install
 mkdir /etc/snort
 mkdir /var/log/snort
 groupadd snort
 useradd –g snort snort
 chown snort:snort /var/log/snort
```

5. Snort Rules Installation

1. Use the snort rules tar file provided on CD1 /snort-2.8.0.1

2. Copy the rules tarball to /etc/snort

```
cd /etc/snort
tar –xvzf snortrules-snapshot-CURRENT.tar.gz
 cp /usr/local/src/snort-2.8.0.1/etc/*.conf* /etc/snort
 cp /usr/local/src/snort-2.8.0.1/etc/*.map  /etc/snort
```

3. Configure the snort.conf file at /etc/snot/snot.conf. Change the following lines less the quotes:

‘var HOME_NET any’ change to:

```
‘var HOME_NET 192.168.100.0/24’    #for IDS1
‘var HOME_NET 192.168.101.0/24’    #for IDS2
‘var EXTERNAL_NET any’ change to:
‘var EXTERNAL_NET !$HOME_NET’
```

Search on the string ‘var RULE_PATH’

Change the line to

```
‘var RULE_PATH /etc/snort/rules’
```

Functional Test

1. Start snort with the following command

```
snort –D –u snort –g snort –c /etc/snort/snort.conf
```

2. Check that eth0 enters promiscuous mode

3. Check /var/log/syslog for Snort initialization completed successfully (pid=some number)

```
tail -f /var/log/syslog
```

4. Run a portscan on the IDS using nmap or any available port scanner.

check /var/log/snort/alerts for a portscan priority 3 message

```
tail -f /var/log/snort/alerts
```

5. If snort works then kill the snort process and continue

End Functional Test

6. Finalize Sensor Installation

Change configuration file and send alerts to XTS400

STOP!

NOTE: Before proceeding with these steps, perform the installation procedures for PostgreSQL including setting up the schema for the snort database.

Functional Test

1. Run the following command:

```
psql -h 192.168.100.130  -U snort -d snort
```

NOTE: The IP address will vary according to your network configuration

2. **<Observe**: should get a snort=> prompt. Exit from psql with \q

End Functional Test

3. Edit snort.conf to send all alerts to the PostgreSQL database

```
cd /etc/snort
```

edit snort.conf

page down to Step #4: Configure output plugins

find the line # output database: alert, postgresql…

NOTE: search for postgresql with your editor’s search feature

Configure the parameters so line looks like this:

```
output database: log, postgresql, user=snort
dbname=snort port=5433 host=192.168.100.130
```

NOTE: your parameters will vary based on your network settings and port settings

Unclassified Postgresql database: port 5433

Secret Postgresql database: port 5434

Functional Test

4. Start snort and check that it initialized correctly:

```
snort –D –u snort –g snort –c /etc/snort/snort.conf
tail –f /var/log/syslog
```

5. Repeat the portscan test. This time the alert should go to the PostgreSQL database.

Login to the PostgreSQL database either remotely from the IDS or locally on the database server.

```
snort=> select * from event;
snort=> select * from signature;
```

Depending on the port scanner used there may be one or two events and one or two signatures

End Functional Test

7. Lokkit Firewall Installation

To give the sensor some protection install a simple iptables based firewall called lokkit

1. Copy lokkit_0.50.22-7.1_i386.deb from CD1 /misc to /usr/local/src

```
dpkg –i lokkit lokkit_0.50.22-7.1_i386.deb
```

2. Run the command *lokkit*

3. Choose medium and customize

Allow port 22 for SSH and the port for the PostgreSQL database you wish to send alerts to. The database port must be added for two-way communication between database server and IDS.

C. XTS400 NETWORK SETUP

This section covers setting up the XTS400 to place two network ports into different security classifications so the IDS can communicate with the PostgreSQL databases. In future work all of the network ports will run at maximum security level.

For this experiment to succeed the following Ethernet ports must be changed to allow the IDS to communicate with the PostgreSQL database.

Device /dev/ether1 (**sl1**:**il0**)

Device /dev/ether2 (**sl5 sc1**:**il0**)

See lab diagram.

Conventions used in this documentation:

For the XTS400 setup documentation all security and integrity levels are in **bold** Arial font and are given as security first then integrity:

Example: **sl0:il0**

Means security level 0 integrity level 0

Example: **min:oss**

Means: lowest possible security level with integrity level set to **il3** with all categories enabled.

<CR> mean carriage return, press the Enter key on the keyboard

1. Configure TCP/IP Parameters for Ether1 and Ether2

NOTE: IP addresses used below are from the Snort MLS testbed configuration, modify as necessary to reflect current topology but it is recommended to use these IP addresses for now.

Login as admin

Set security and integrity levels - **min:max**

```
SAK

Enter command?                          tcpip_edit

Enter editor request?                   add

Enter TCP/IP daemon name?               tcpip_unclass

Enter TCPIP/IP daemon description? Unclass TCP/IP connection


Enter domain name?                      <CR>

Enter host name?                        mlsserver

Will this daemon support IPv4?                  Y

Will this daemon support IPv6?                  <CR>

Treat all subnets in network as local[N]        <CR>

Shutdown on failure[N]                          <CR>

Use default TCP minimum retransmission timeout[Y] <CR>

Enable IPv4 forwarding[N]                       <CR>

Send IPv4 redirects[Y]                          <CR>

Forward IPv4 source routed packets[N]           <CR>

Accept IPv4 source routed packets[N]            <CR>

Drop ICMP4 redirects[Y]                         <CR>

Reply to ICMP4 netmask requests[N}              <CR>

Use randomly generated IPv4 datagram IDs[N]     <CR>

Add the network interface configuration[Y]      <CR>

Enter TCP/IP device name?                       /dev/ether1

Is IPv4 active on this interface?[Y]            <CR>
```

Enter the IPv4 address in the format A.B.C.D *192.168.100.130*

Enter the IPv4 network mask in the format A.B.C.D *255.255.255.0*

Use default IPv4 broadcast address?[Y] <CR>

<observe: Network interface /dev/ether1 added.>

Add another network interface entry[N] N

NOTE: this is the interface for the other IDS on the secret network /dev/ether2

add

Enter TCP/IP daemon name? *tcpip_secret*

Enter TCPIP/IP daemon description? *Secret TCP/IP connection*

Enter domain name? <CR>

Enter host name? *mlsserver*

Will this daemon support IPv4? Y

Will this daemon support IPv6? <CR>

Treat all subnets in network as local[N] <CR>

Shutdown on failure[N] <CR>

Use default TCP minimum retransmission timeout[Y] <CR>

Enable IPv4 forwarding[N] <CR>

Send IPv4 redirects[Y] <CR>

Forward IPv4 source routed packets[N] <CR>

Accept IPv4 source routed packets[N] <CR>

Drop ICMP4 redirects[Y] <CR>

Reply to ICMP4 netmask requests[N} <CR>

Use randomly generated IPv4 datagram IDs[N] <CR>

Add the network interface configuration[Y]   <CR>

Enter TCP/IP device name?   */dev/ether2*

Is IPv4 active on this interface?[Y]   <CR>

Enter the IPv4 address in the format A.B.C.D *192.168.101.130*

Enter the IPv4 network mask in the format A.B.C.D   *255.255.255.0*

Use default IPv4 broadcast address?[Y]   <CR>

<observe: Network interface /dev/ether1 added.>

Add another network interface entry[N]   N

Add the route configuration[N]   <CR>

Add the resolver configuration[N]   <CR>

<observe: TCP/IP Daemon tcp_unclass added and tcp_secret added>

exit

NOTE: remove all references in the tcpip_mls to all Ethernet devices except /dev/ether0

2. Configure the Daemon for tcp_unclass and tcp_secret

SAK - level should still be **min:max**

Enter command?   *daemon_edit*

Enter editor request?   *add*

Enter daemon name?   *tcpip_unclass*

Enter the command line for the daemon program?   *tcpip*

Enter the arguments for the program?   <CR>

Enter daemon environment setting?   <CR>

Start daemon at startup?   Y

Is the daemon a High Integrity program file?   Y

Will this daemon control a device ? N

Enter daemon security level and categories **sl1**

Enter daemon integrity level and categories **il0**

Use default daemon priority [-1] <CR>

Enter user name? *`network`*

Enter group name? *`network`*

Display current starting order before setting start index[N] <CR>

Enter daemon start index? 2

NOTE:, tcpip daemons must start first so if there are other daemons put the tcpip daemon at the beginning of the order before other non tcpip daemons

Enter the delay interval in seconds before starting the daemon? 0

Enter the delay interval in seconds while stopping the daemon? 0

Enter daemon description? *`TCP_IP daemon for unclassified IDS`*

<observe: Daemon entry added.

NOTE: repeat the process to add a daemon for the tcp_secret network

Enter editor request? *`add`*

Enter daemon name? *`tcpip_secret`*

Enter the command line for the daemon program? *`tcpip`*

Enter the arguments for the program? <CR>

Enter daemon environment setting? <CR>

Start daemon at startup? Y

Is the daemon a High Integrity program file? Y

Will this daemon control a device ? N

Enter daemon security level and categories **sl5 sc1**

Enter daemon integrity level and categories **il0**

Use default daemon priority [-1] <CR>

Enter user name? *network*

Enter group name? *network*

Display current starting order before setting start index[N] <CR>

Enter daemon start index? 3

NOTE:, tcpip daemons must start first so if there are other daemons put the tcpip daemon at the beginning of the order before other non tcpip daemons

Enter the delay interval in seconds before starting the daemon? 0

Enter the delay interval in seconds while stopping the daemon? 0

Enter daemon description? *TCP_IP daemon for secret IDS*

<observe: Daemon entry added.

exit

3. Configure the device (SDA) for /dev/ether1 and /dev/ether2

NOTE: change security and integrity level to **max:max**

SAK

Enter command? *sda*

Enter device? */dev/ether1*

Enter new device security level and categories? **sl1**

Enter new device integrity level and categories? **il0**

Modify discretionary access? Y

Enter device modes for owner? *rw*

Enter user name for specific permission? <CR>

Enter device modes for group? *rw*

Enter group name for specific permission? <CR>

Enter device modes for others? *rw*

(VERIFY user & group is SYSTEM)

Is access correct? Y

NOTE: repeat for device /dev/ether2

SAK

Enter command? *sda*

Enter device? */dev/ether2*

Enter new device security level and categories? **sl5 sc1**

Enter new device integrity level and categories? **il0**

Modify discretionary access? Y

Enter device modes for owner? *rw*

Enter user name for specific permission? <CR>

Enter device modes for group? *rw*

Enter group name for specific permission? <CR>

Enter device modes for others? *rw*

(VERIFY user & group is SYSTEM)

Is access correct? <CR>

NOTE: If ‘startup’ has been performed then proceed to step 4 to start the daemons. If the tcpip_mls daemon was modified then restart that daemon before proceeding to step 4.

4. Start the daemons

SAK

Enter command? *start_daemon*

Enter daemon name? *tcpip_unclass*

<observe: daemon started successfully>

Enter daemon name? *tcpip_secret*

<observe: daemon started successfully>

Enter daemon name? <CR>

<observe: Start_daemon complete>

Functional Test

5. Testing IP interfaces.

Set security and integrity level to **sl1**:**il0**

SAK

run

At command prompt type: *ifconfig*

```
<observe: lo0        inet    addr   127.0.0.1        netmask 255.0.0.0
/dev/ether1   inet    addr   192.168.100.130    netmask 255.255.255.0
```

Repeat for security and integrity level of **sl5 sc1**:**il0**

End Functional Test

D. INSTALLING POSTGRESQL 7.4.18 ON STOP 6.3

This section will cover installing the open source database PostgreSQL version 7.4.18 on the STOP 6.3 operating system. Three data areas will be initialized, one just for regular testing, one data area for Snort's unclassified database and one data area for Snort's secret database.

Postgres 7.4.18 Pre-Installation Instructions

To operate Postgresql and installed databases for the Snort sensors two users need to be created. User postgres will run the postmaster database process. User snort is required to make the database snort. Postgres 7.4.18

does not have a function to give ownership to an entire database, only tables. In order for the remote user snort coming in from the snort sensor to utilize the database the user snort must make the database snort.

<u>NOTE</u>: copying files from CD to the STOP 6.3 file system requires the use of the 'cdtool'. An example of using the cdtool is:

```
cdtool cp /dev/cdrom /postgresql-7.4.18/postgresql-7.4.18.tar.gz /usr/local/src/postgresql-7.4.18.tar.gz
```

1. Create Groups Named Postgres and Snort

Perform the following steps as admin at (max:max)

SAK

Enter command?	*ga_edit*
?	*add*
Enter group name?	*postgres*
Enter group number (1 - X)	<u>NOTE:</u> Use the last number
Enter user list request (a, d, l or q)	*q*

<Observe: Group entry added

*Repeat steps starting with 'add' for group snort

exit

2. Create Users Named Postgres and Snort

Perform the following steps as admin with applicable security and integrity levels to run the ua_edit process. (**max:max**)

<u>NOTE:</u> The function of user 'postgres' is to own the postmaster process and the database areas. The function of user snort is to create the snort databases. Neither user needs to have the capabilities to remotely logon to the XTS400 via the TPE therefore, they should not be entire setup as normal users.

SAK

Enter command?	*ua_edit*
	add
Enter user name?	*postgres*

Enter user number?	Select the next available number
Enter default group?	*postgres*
Add user to group?	Y
Enter command processor?	*/bin/sh*
Enter home directory?	*/home/postgres*
Enter user max security level and categories?	**max**
Enter user max integrity level and categories?	**max**
Enter user default security level and categories:	**min**
Enter user default integrity level and categories:	**oss**
Down grade allowed?	n
Upgrade allowed?	n
View optional when downgrading?	n
Change user password allowed?	y
Disconnect allowed?	y
Kill, ikill allowed?	y
Run allowed?	y
Set group allowed?	n
Set level, change default level allowed?	y
Shutdown allowed?	n
Unmarked printed output allowed?	n
multiple logins allowed?	y
change other users' passwords allowed?	n

repeat ua_edit and create a user called snort with group snort and home= /home/snort. Duplicate the other settings

Change the password of each user using the cup command

SAK

Enter command?	cup
Enter user name?	postgres
Force user to change password at next authentication [N]	<CR>

NOTE: Password length must be 6 - 16 characters in length and contain numerals and special characters.

Enter new user password?   xts400

Enter new user password again?   xts400

<Observe: User password changed

Repeat for user snort.

```
exit
```

After making the groups and users the group and passwd file needs to be updated.

Set security and integrity level: **min:oss**

SAK

Enter command? *run*, or *reattach* to previous session at min:oss

From the shell prompt run the following commands:

```
/xts/untrusted/bin/xtsmkgroup > /etc/group
chmod 644 /etc/group

/xts/untrusted/bin/xtsmkpasswd > /etc/passwd
chmod 644 /etc/passwd
```

3. PostgreSQL 7.4.18 Installation

Perform following steps as **admin** at level (**min:oss**)

1. Make directory /usr/local/src

```
    mkdir /usr/local/src
    cd /usr/local/src
```

2. Copy postgresql-base-7.4.18.tar.gz from CD1 /postgresql-7.4.18 to /usr/local/src

```
cdtool cp /dev/cdrom /postgres-7.4.18/postgresql-base-
7.4.18.tar.gz /usr/local/src/postgresql-7.4.18.tar.gz
tar -xvzf /usr/local/src/postgresql-base-7.4.18.tar.gz

cd /usr/local/src/postgresql-7.4.18
```

<u>NOTE</u>: Compiling Postgresql requires gmake > 3.76.1

gmake --version = 3.8 on XTS400

ISO/ANSI C compiler such as GCC

3. Run configure script

```
     ./configure --without-readline --without-zlib
```

NOTE: default installation is /usr/local/pgsql

All other files install to sub-directories under /pgsql

Use *./configure --help* for other alternatives

NOTE: As of this installation <5/10/2008> STOP 6.3.1 did not have readline library or zlib library installed, thus reason for the extra parameters for the ./configure command

NOTE: ignore the errors related to Flex and Bison, they are only required if building from CVS or changing any of the scanner definition files.

4. Start build with gmake:

```
gmake
```

<Observe: final output is "All of PostgreSQL successfully made. Ready to install">

5. (optional) Regression Tests:

as a non-privileged user:

```
gmake check
```

6. Install the files:

```
gmake install
```

<Observe: PostgreSQL installation complete>

4. Post-installation Setup

Perform as user admin level (min:oss)

1. Update Environment Variables

add to /etc/profile

```
PATH=/usr/local/pgsql/bin:$PATH
MANPATH=/usr/local/pgsql/man:$MANPATH
```

2. Initialize a database storage area on disk:

NOTE: this storage area will be at **min:oss**

```
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
```

Log out of XTS400 as admin and log in as **postgres**

Start untrusted environment at **min:oss**

SAK

```
run
initdb -D /usr/local/pgsql/data
```

```
<Observe:
initdb -D /usr/local/pgsql/data…
>
```

3. Starting and stopping the database server as user postgres:

Use the pg_ctl script as user postgres to start the databases

type `pg_ctl --help` for options

Start: `pg_ctl start -D  /usr/local/pgsql/data -l serverlog`

Stop: `pg_ctl stop -D /usr/local/pgsql/data -m fast`

Functional Test

4. Test the installation by making a database called mydb then login as postgres. Postgressql 7.4.18 requries logging in with username and database unless the database is the same name as the user.

```
cd /usr/local/pgsql/data
pg_ctl start -D /usr/local/pgsql/data -l logfile
createdb mydb
psql -U postgres -d mydb
```

use \q to quit

Stop the database:

```
pg_ctl stop –D /usr/local/pgsql/data –m fast
```

Logout from the untrusted environment

Logout as **postgres**

This completes setting up postgresql to run at level **min:oss**

5. Configure PostgreSQL to Operate at Multiple Security Levels

Perform the following steps as **admin** unless noted.

1. Create two data locations for starting up two more instances of PostgreSQL

Set security and integrity levels - **min:max**

SAK

```
Enter command?                      fsm
                                    mkdir
Enter the directory to create?      /var/postgresql
Should this be a deflection directory [N]    <CR>
<Observe: Directory added
                                             change
Enter pathname?                              /var/postgresql
Modify access level [N]?                              Y
Enter new object security level and categories?       sl0
Enter new object integrity level and categories?       il3
Is the level correct [Y]                              <CR>
Modify discretionary access [N]                       Y
Enter new owner name?                                 admin
Enter new group name?                                 stop
Enter object modes for owner?                         rwx
Enter user name for specific permission?              postgres
Enter object modes for specific user?                 rx
Enter user name for specific permission?              snort
Enter object modes for specific user?                 rx
Enter user name for specific permission?              <CR>
Enter object modes for group?                         <CR>
Enter group name for specific permission?             <CR>
Enter object modes for others?                        <CR>
Display the object [N]                                <CR>
Okay to change [Y]                                    <CR>
```

NOTE: User postgres and snort should have rx access to /var/postgresql

```
                                                      mkdir
Enter the directory to create? /var/postgresql/unclass_data
Should this be a deflection directory [N]             <CR>
```

<Observe: Directory created>

```
                                                   change
Enter pathname?          /var/postgresql/unclass_data
Modify access level?                               Y
Enter new object security level and categories?    sl1
Enter new object integrity level and categories?    il0
Is the level correct [Y]                           <CR>
Modify discetionary access [N]                      Y
Enter new owner name?                              postgres
Enter new group name?                              postgres
Enter object modes for owner?                       rwx
Enter user name for specific permission?           snort
Enter object modes for specific user?              rwx
Enter user name for specific permission?           admin
Enter object modes for specific user?              rwx
Enter user name for specific permission             <CR>
Enter object modes for group?                      <CR>
Enter group name for specific permission?          <CR>
Enter object modes for others?                     <CR>
Display the object [N]                             <CR>
Okay to change [Y]                                 <CR>
```

NOTE: only users postgres, snort, and admin should have rwx access

```
Access changed
                                                   mkdir
Enter the directory to create?  /var/postgresql/secret_data
Should this be a deflection directory[N]           <CR>
                                                   change
Enter pathname?          /var/postgresql/secret_data
Modify access level?                               Y
Enter new object security level and categories?    sl5 sc1
```

Enter new object integrity level and categories?	**il0**
Is the level correct [Y]	<CR>
Modify discetionary access [N]	Y
Enter new owner name?	*postgres*
Enter new group name?	*postgres*
Enter object modes for owner?	*rwx*
Enter user name for specific permission?	*snort*
Enter object modes for specific user?	*rwx*
Enter user name for specific permission?	*admin*
Enter object modes for specific user?	*rwx*
Enter user name for specific permission	<CR>
Enter object modes for group?	<CR>
Enter group name for specific permission?	<CR>
Enter object modes for others?	<CR>
Display the object [N]	<CR>
Okay to change [Y]	<CR>

NOTE: only users postgres, snort, and admin should have rwx access

Access changed

exit

logout admin

2. Initialize the database areas

Log into XTS400 as user **postgres**

Set security and integrity levels to **sl1**:**il0**

SAK

run

cd /var/postgresql/unclass_data

initdb -D /var/postgresql/unclass_data

<Observe that the output is the same as step 2 Post-Installation Setup

Run command *ls*

<Observe that files and directories were created>

Edit postgresql.conf

Remove # from line `tcpip_socket = false` and change the line to read: `tcpip_socket = true`

Remove # from `line port = 5432` and change the line to read

```
        port = 5433
```

Save the file

Edit pg_hba.conf

Page down to the end of the file

Remove the IPv6 and local host entries NOTE: The MYSEA host file does not use local host entries and localhost is not required.

Append the following to the proper headers:

TYPE	`host`
DATABASE	`snort`
USER	`snort`
IP_ADDRESS	`192.168.100.0`
MASK	`255.255.255.0`
AUTH_TYPE	`MD5`

Save the file

Start the database

```
pg_ctl start -D /var/postgresql/unclass_data -l
logfile
```

Create the PostgreSQL database user snort so XTS400 user snort can create the database snort

Run the script `createuser -p 5433 snort`

Shall the new user be allowed to create databases? y

Shall the new user be allowed to create more new users? n

<Observe: CREATE USER>

Alter user snort’s database password

```
psql -p 5433 -U postgres -d template1
alter user snort password ‘mysea’;
\q
```

Stop the database

```
pg_ctl stop -D /var/postgresql/unclass_data -m fast
```

REPEAT for data area /var/postgresql/secret_data

Log out of session

Change security and integrity levels to **sl5 sc1**:**il0**

SAK

```
run
cd /var/postgresql/secret_data
initdb -D /var/postgresql/secret_data
```

<*Observe* that the output is the same as step 2 Post-Installation Setup

Run command *`ls`*

<*Observe* that files and directories were created>

Edit postgresql.conf

Remove # from line tcpip_socket = false and change the line to read

```
tcpip_socket = true
```

Remove # from line port = 5432 and change the line to read

```
        port = 5434
```

Save the file

Edit pg_hba.conf

Page down to the end of the file

Remove the IPv6 and localhost enteries. <u>NOTE:</u> The MYSEA host file does not use local host entries and localhost is not required.

Append the following entries to the proper headers

TYPE	`host`
DATABASE	`snort`
USER	`snort`
IP_ADDRESS	`192.168.101.0`
MASK	`255.255.255.0`
AUTH_TYPE	`MD5`

Save the file

Start the database

```
pg_ctl start -D /var/postgresql/secret_data -l
```

```
logfile
```

Create the postgresql database user snort so XTS400 user snort can create the database snort.

Run the script `createuser -p 5434 snort`

Shall the new user be allowed to create databases?   y

Shall the new user be allowed to create more new users?   n

<*Observe*: CREATE USER>

Alter user snort’s password

```
psql -p 5433 -U postgres -d template1
alter user snort password ‘mysea’;
\q
```

Stop the database

```
pg_ctl stop -D /var/postgresql/secret_data -m fast
```

3. Create two daemons for starting the new postgresql database areas

log user **postgres** out of the system

log into XTS as **admin**

set security and integrity levels - **min:max**

SAK

Enter command?   *tp_edit*

type in *cd* then then enter to change to /system directory

                                                              add

Enter program name?   *pg_ctl*

Enter pathname?

```
/usr/local/pgsql/bin/pg_ctl
```

Enter maximum integrity?   <CR>

Enter minimum integrity?   <CR>

Assign privileges [N]   Y

Only assign Y to the following options:

Enable "Set owner/group" privilege

Modify discretionary access [N]   Y

Enter new owner name? *postgres*

Enter new group name? *postgres*

Enter object modes for owner? *rwx*

Enter user name for specific permission? *snort*

Enter object modes for user? *rwx*

Enter group name for specific permission? *snort*

Enter object modes for group? *rwx*

Enter object modes for others? *none*

SAK - level should still be **min:max**

Enter command? *daemon_edit*
add

Enter daemon name? *psql_unclass*

Enter the command line for the daemon program? *pg_ctl*

Enter the arguments for the program?

start -D /var/postgresql/unclass_data -l
/var/postgresql/unclass_data/logfile

Enter daemon environment setting? <CR>

Start daemon at startup? y

Is the daemon a High Integrity program file? y

Will this daemon control a device? n

Enter daemon security level and categories? **sl1**

Enter daemon integrity level and categories? **il0**

Use default daemon priority [-1] (strongly recommended) [Y] <CR>

Enter user name? *postgres*

Enter group name? *postgres*

Display current starting order before setting start index [N] <CR>

Enter daemon start index (CR for end of list) (1-7) <CR>

Enter the delay interval in seconds before starting the daemon? 4

Enter the delay interval in seconds while stopping the daemon? 4

Enter daemon description? *start postgres at level* *unclassified*

REPEAT to create daemon to start postgresql at secret

add

Enter daemon name? *psql_secret*

Enter the command line for the daemon program? *pg_ctl*

Enter the arguments for the program?

start -D /var/postgresql/secret_data -l /var/postgresql/secret_data/logfile

Enter daemon environment setting? <CR>

Start daemon at startup? y

Is the daemon a High Integrity program file? y

Will this daemon control a device? n

Enter daemon security level and categories? **sl5 sc1**

Enter daemon integrity level and categories? **il0**

Use default daemon priority [-1] (strongly recommended) [Y] <CR>

Enter user name? *postgres*

Enter group name? *postgres*

Display current starting order before setting start index [N] <CR>

Enter daemon start index (CR for end of list) (1-7) <CR>

Enter the delay interval in seconds before starting the daemon? 4

Enter the delay interval in seconds while stopping the daemon? 4

Enter daemon description? *Start postgres at level secret*

exit

Start both daemons:

SAK

Enter command? *start_daemon*

Enter daemon name? *psql_unclass*

<*Observe*: Daemon psql_unclass started successfully>

Enter daemon name? *psql_secret*

<*Observe*: Daemon psql_secret started successfully>

NOTE: This is not a guarantee that the databases are running. To verify the databases started properly, either remotely login with an available psql client or login locally.

Functional Test

Example:

Change security and integrity level to **sl1**:**il0**

SAK

run

psql -p 5433 -U postgres -d template1

<*Observe*: Welcome to psql, the PostgreSQL interactive terminal>

End Functional Test

4. Create the snort database

Log in to XTS as user **snort**

Change security and integrity level to **sl1**:**il0**

SAK

run

cd /var/postgresql/unclass_data

Run script *createdb -p 5433 snort*

<*Observe*: CREATE DATABASE>

Run the schema update using create_postgresql from CD1 /snort-2.8.0.1

SAK

<Change security and integrity level to max:max>

SAK

Enter command? *sda*

Enter device? */dev/cdrom*

Enter new device security level and categories? **sl1**

Enter new device integrity level and categories? **il0**

Modify discretionary access [N] <CR>

Is access correct [Y] <CR>

<Observe: Device access has been set.

SAK

<Change security and integrity level to **sl1:il0**>

SAK

Enter command *reattach*

Enter family number? 1

cdtool cp /dev/cdrom /snort-2.8.0.1/create_postgresql ./create_postgresql

psql -p 5433 < ./create_postgresql

Log out of untrusted environment

Change security and integrity level to **sl5 sc1**:**il0**

SAK

run

cd /var/postgresql/secret_data

run script createdb -p 5434 snort

<Observe: CREATE DATABASE>

run the schema update using create_postgresql from CD1 /snort-2.8.0.1

SAK

<Change security and integrity level to max:max>

SAK

Enter command? *sda*

Enter device? */dev/cdrom*

Enter new device security level and categories? **sl5 sc1**

Enter new device integrity level and categories? **il0**

Modify discretionary access [N] <CR>

Is access correct [Y] <CR>

<Observe: Device access has been set.

SAK

<Change security and integrity level to **sl5 sc1:il0**>

SAK

Enter command *reattach*

Enter family number? 2

```
cdtool cp /dev/cdrom /snort-2.8.0.1/create_postgresql ./create_postgresql
```

```
psql -p 5434 < ./create_postgresql
```

Log out of untrusted environment

Reset CD-ROM back to **min:oss**

SAK

<change security and integrity level to max:max>

SAK

Enter command? *sda*

Enter device? */dev/cdrom*

Enter new device security level and categories? **min**

Enter new device integrity level and categories? **oss**

Modify discretionary access [N] <CR>

Is access correct [Y] <CR>

<Observe: Device access has been set.

Log out user snort from XTS400

5. Perform functional test if the sensors are finished:

Functional Test

Login to PostgresSQL database from sensor IDS1

```
psql -h 192.168.100.130 -p 5433 -U snort -d snort
```

Run a query

```
select * from event;
```

Should get 0 rows.

NOTE: If the output is access denied the database is not owned by snort

Login to PostgreSQL database from sensor IDS2

```
psql -h 192.168.101.130 -p 5434 -U snort -d snort
```

Run a query

```
select * from event;
```

Should get 0 rows.

NOTE: If the output is access denied the database is not owned by snort

End Functional Test

6. Return to Installing Debian snort sensor documentation and complete the steps starting from Step 6. Finalize Sensor Installation.

E. INSTALLING BASE WEB APPLICATION ON STOP 6.3

This section will cover integrating PHP4.3 with the existing httpd so the BASE web application functions with httpd. The MYSEA environment must already be installed for these procedures to work.

NOTE: This will be a static install. See the INSTALL file in the php4 directory

NOTE: some components of php4 require pthread support. Pthreads was not implemented in STOP 6.3 but the pthread.h, libpthread.a and libpthread.so files still exist under /usr/include and /usr/lib respectively. Having these files in this location causes the configure process to crash on a check looking for pthread_cflags. Rename those files to pthread.h.old, libpthread.a.old and libpthread.so.old.

NOTE: copying files from CD to the STOP 6.3 file system requires the use of the ‘cdtool’. An example of using the cdtool is: *cdtool cp /dev/cdrom /postgres-7.4.18/postgresql-7.4.18.tar.gz /usr/local/src/postgresql-7.4.18.tar.gz*

For the XTS400 setup documentation all security and integrity levels are in **bold** Arial font and are given as security first then integrity:

Example: **sl0:il0**

Means security level 0 integrity level 0

Example: **min:oss**

Means: lowest possible security level with integrity level set to **il3** with all categories enabled.

1. PHP 4.3.11 Installation

Perform the flex and bison install as **admin <min:oss>**

NOTE: php4 requires flex and suggests bison

Copy flex-2.5.4a-26.i386.rpm from CD1 /misc /usr/local/src

```
cdtool cp /dev/cdrom /misc/flex-2.5.4a-26.i386.rpm
/usr/local/src/flex-2.5.4a-26.i386.rpm
```

Copy bison-1.35-4.i386.rpm from CD1 /misc to /usr/local/src

```
cdtool cp /dev/cdrom /misc/bison-1.35-4.i386.rpm
/usr/local/src/bison-1.35-4.i386.rpm
rpm -iv flex-2.5.4a-26.i386.rpm
rpm -iv bison-1.35-4.i386.rpm
```

Perform the php4 install as **admin** <min:**il3**>

Copy php-4.3.11.tar.tar from CD1 /php-4.3.11 to /usr/local/mysea

SAK

<Change security and integrity level to max:max>

SAK

```
Enter command?                                          sda
Enter device?                                           /dev/cdrom
Enter new device security level and categories?           min
Enter new device integrity level and categories?          il3
Modify discretionary access [N]          <CR>
Is access correct [Y]                    <CR>
```

<Observe: Device access has been set.

SAK

<Change security and integrity level **min:il3**>

SAK

```
Enter command?          reattach
Enter family number?    3
```

```
cdtool cp /dev/cdrom /php-4.3.11/php-4.3.11.tar.tar
/usr/local/mysea
tar -xvzf php-4.3.11.tar.tar
mv php-4.3.11 php4
cd php4
```

NOTE: this install will be without graphics support for BASE which is just the ability to make charts and graphs.

```
cp php.ini-dist /home/http/conf/php.ini
./configure --without-mysql --without-pear --with-
apache=/usr/local/mysea/apache --with-
pgsql=/usr/local/pgsql --with-config-file-
path=/home/http/conf
```

Edit the Makefile

Search for EXTRA_INCLUDES =

Add this line: `-I/usr/local/mysea/include`

NOTE: remember the Makefile statements are tab delimited (separators)

```
make
make install prefix=/usr/local/mysea
```

<Observe:

```
Installing PHP SAPI module: apache
Installing PHP CLI binary:  /usr/local/mysea/bin
Installing PHP CLI man page: /usr/local/mysea/man/man1/
Installing build environment: /usr/local/mysea/lib/php/build
Installing header files: /usr/local/mysea/include/php/
Installing helper programs: /usr/local/mysea/bin
    program: phpsize
    program: php-config
    program: phpextdist
```

2. Recompile Httpd to Include PHP Module

```
cd /usr/local/mysea/apache/src
```

Edit Configuration file

NOTE file attributes may need to be changed to rw for **admin** (*chmod 740 Configuration*)

Page down to the end of the file

Add this line: `AddModule modules/php4/libphp4.a`

Save the file and recompile

```
./Configure
make
```

<Observe: there are no errors, just a warning about the use of mktemp, which can be ignored>

<Expect: the size of httpd should now be approximately 5.8MB (5808437) in size. Verify with *ls -l httpd*>

3. Httpd Installation

Save the old httpd file under /usr/local/mysea/bin by renaming it or copy it off to a safe place and copy the new file into mysea/bin

```
cp /usr/local/mysea/apache/src/httpd /usr/local/mysea/bin
cd /usr/local/mysea/bin
chmod 555 httpd
cd /home/http/conf
```

NOTE: might have to set write attribute on httpd.conf for **admin**

```
    chmod 644 httpd.conf
```

Edit /home/http/conf/httpd.conf

Add the following lines to the httpd.conf file or uncomment the lines if they exist:

```
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps
```

chmod 444 httpd.conf if you did chmod 644 httpd.conf

4. Test PHP

Functional Test

```
mkdir /home/http/htdocs/ids_demo
chmod 755 /home/http/htdocs/ids_demo
```

Create a test script called test.php in /home/http/htdocs/ids_demo

Add the following to the script:

```
<?php phpinfo();?>
```

Save the file and exit the untrusted environment

From a mysea workstation that has TCBE installed:

Start the TCBE

Click on SAR

Enter user name: *mdemo1*

Enter password: *password*

Click on SAR

Enter command: *run*

Open an Internet browser

In the URL address bar enter:

http://192.168.0.130/ids_demo/test.php

<Observe: a PHP Version 4.3.11web page is generated>

End Functional Test

5. BASE Installation

Since there are two IDS one unclassified and one secret there needs to be two BASE installs, one unclassified, and one secret.

Copy base-1.4.0.tar.gz from CD1 /base-1.4.0 to /home/http/htdocs/ids_demo/

```
cdtool cp /dev/cdrom /base-1.4.0/base-1.4.0.tar.gz
/home/http/htdocs/ids_demo/base-1.4.0.tar.gz
```

Exit untrusted environment

Set security and integrity level **min**:**il3**

SAK

Enter command? *fsm*

? *mkdir*

Enter the directory to create?

/home/http/htdocs/ids_demo/base_unclass

Should this be a deflection directory [N] <CR>

<Observe: Directory created>

? *change*

Enter pathname?

/home/http/htdocs/ids_demo/base_unclass

Modify access level [N] Y

Enter new object security level and categories? **sl1**

Enter new object integrity level and categories? **il0**

Is the level correct [Y] <CR>

Modify discretionary access [N] <CR>

Display the object [N] <CR>

Okay to change [Y] <CR>

**REPEAT to make directory for base_secret

? *mkdir*

Enter the directory to create?

/home/http/htdocs/ids_demo/base_secret

Should this be a deflection directory [N] <CR>

<Observe: Directory created>

? *change*

Enter pathname? */home/http/htdocs/ids_demo/base_secret*

Modify access level [N] Y

Enter new object security level and categories? **sl5 sc1**

Enter new object integrity level and categories? **il0**

Is the level correct [Y] <CR>

Modify discretionary access [N] <CR>

Display the object [N] <CR>

Okay to change [Y] <CR>

```
exit
```

Change security and integrity level **sl1**:**il0**

SAK

```
run
cd /home/http/htdocs/ids_demo
tar -C base_unclass -xvzf base-1.4.0.tar.gz
cd base_unclass/base-1.4.0
mv * /home/http/htdocs/ids_demo/base_unclass
chmod 777 /home/http/htdocs/ids_demo/base_unclass
```

NOTE the setup performed in step 7 requires rwx access. Will change back later

Exit the untrusted environment

Change security and integrity level **sl5 sc1**:**il0**

SAK

```
run
cd /home/http/htdocs/ids_demo
tar -C base_secret -xvzf base-1.4.0.tar.gz
cd base_secret/base-1.4.0
mv * /home/http/htdocs/ids_demo/base_secret
chmod 777 /home/http/htdocs/ids_demo/base_secret
```

NOTE the setup performed in step 8 requires rwx access to write the base_conf.php file. Will change back later

exit untrusted environment

Change security and integrity level min:**il3**

SAK

```
run
```

copy adodb498.gz from CD1 /base-1.4.0 to /usr/local/mysea

```
cdtool cp /dev/cdrom /base-1.4.0/adodb498.gz /usr/local/mysea/adodb498.gz
```

```
cd /usr/local/mysea
tar -xvzf adodb498.gz
```

<Observe: adodb directory is created. There is nothing to install but the scripts running in base_unclass and base_secret will need access to these files>

6. Unclassified BASE Website Installation

Start a session on a MYSEA client at SIM_UNCLASSIFIED

Using Internet Explorer enter this address into the URL address bar:

```
http://192.168.0.130/ids_demo/base_unclass/setup/index.php
```

NOTE: if you get access denied check the directory attributes and chmod to 777 for now

Click on continue hyperlink

Step 1 of 5

Path to ADODB: `/usr/local/mysea/adodb`

Click on Submit Query button

Step 2 of 5

Pick a Database type: `PostGRES`

Database Name: `snort`

Database Host: leave blank

Database port: `5433`

Database user name: `snort`

Database password: mysea

Skip the Use Archive Database settings

Click on Submit Query button

Step 3 of 5

Click the box next to Use Authentication System

Admin User name: `snort`

NOTE: you can use any name you want

Password: `mysea`

Full Name: *snort*

Step 4 of 5

Click on Create BASE AG button

Click on 'Now continue to step 5... hyperlink (middle page)

Login with user and password created in step 3 of 5

When you come back to this site, the main page is base_main.php

http://192.168.0.130/ids_demo/base_unclass/base_main.php

NOTE: don't forget to chmod 755 base_unclass

Cosmetic changes:

cd /home/http/htdocs/ids_demo/base_unclass/languages

Edit english.lang.php

Search for _TITLE

Add the word UNCLASS before and after the title in single quotes

bug_fix: minor

When logging out: Fatal error: call to function base_header() in

/.../base_logout.php on line 23

FIX: change base_header to header

7. Secret BASE Website Installation

Start a session on a MYSEA client at SIM_SECRET

Using Internet Explorer, enter this address into the URL address bar:

http://192.168.0.130/ids_demo/base_secret/setup/index.php

NOTE: if you get access denied check the directory attributes and chmod to 777 for now

Click on continue

Step 1 of 5

Path to ADODB: */usr/local/mysea/adodb*

Click on Submit Query button

Step 2 of 5

Pick a Database type: PostGRES

Database Name: *snort*

Database Host: leave blank

Database port: *5434*

Database user name: *snort*

Database password: leave blank

Skip the Use Archive Database settings

Click on Submit Query

Step 3 of 5

Click the box next to Use Authentication System

Admin User name: *snort*

NOTE: you can use any name you want

Password: *mysea*

Full Name: *snort*

Step 4 of 5

Click on Create BASE AG button

Click on 'Now continue to step 5... hyperlink (middle page)

Login with user and password created in step 3 of 5

To return to this web site use the main page: base_main.php

```
http://192.168.0.130/ids_demo/base_secret/base_main.php
```

NOTE: don't forget to chmod 755 base_secret

Cosmetic change:

```
cd /home/http/htdocs/ids_demo/base_secret/languages
```

Edit english.lang.php

Search for _TITLE

Add the word SECRET before and after the title in single quotes

bug_fix: minor

When logging out: Fatal error: call to function base_header() in /.../base_logout.php on line 23

FIX: change base_header to header

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C FUNCTIONAL TEST PLAN

This section covers a functional test of the IDS architecture for the Linux Red Hat 8 environment and the XTS400 environment. The purpose of this functional test is to test all of the IDS components as one functional unit. When the alert mechanism on the IDS sensor is triggered data should move from the IDS sensor via the PostgreSQL client to the PostgreSQL database. Data should then be available for display using the BASE web application. The alert mechanism will be triggered using an IDS false positive test generator called ‘IDSWakeup’.

IDS Wakeup sends several packets to a destination IP address with each packet containing data that is interpreted as malicious data. Numerous unique alerts are generated depending on the rule sets enabled on the sensor.

A. PRELIMINARY SETUP FOR FUNCTIONAL TESTING

1. IDSWakeup Installation

1. Build or clone another Debian 4.0 machine and name it Attacker. If the choice is to build a new machine use the Installing Snort on Debian 4.0 instructions from Appendix A. Perform the installation except for installing snort, snort rules, and lokkit.

2. Copy the three files to any directory. Change to that directory and install all three packages in the following order

```
dpkg -i hping2_2.rc3-4_i386.deb
dpkg -i libnet0_1.0.2a-7_i386.deb
dpkg -i idswakeup_1.0.4_i386.deb
```

If the Attacker and IDS sensors are on VMWare and the VMWare machines are using bridged networking on one physical network port, both IDS will see the traffic. If the physical machine has two network ports use separate bridges for each IDS sensor. Remember to switch the bridge on the Attacker when switching from unclass network to secret network.

B. RED HAT 8 IDS ARCHITECTURE FUNCTIONAL TEST

1. Before starting IDSWakeup verify database and web site components are available.

2. Edit the Snort configuration file /etc/snort/snort.conf on IDS1 and IDS2 to send alerts to the PostgreSQL database on the Red Hat 8 Linux system.

Settings for IDS1:

```
output database: log, postgresql, user=snort
dbname=snort password=mysea host=192.168.100.3
port=5433
```

Settings for IDS2:

```
output datagase: log, postgresql, user=snort
dbname=snort password=mysea host=192.168.101.3
port=5434
```

3. Stop any running snort processes on each IDS:

```
ps aux | grep snort
```

```
kill <pid number>
```

These tests will run Snort without using the daemon mode. This will allow for collecting statistics that will be used to compare with the statistics viewed on the BASE web site. Start the Snort process on each IDS sensor

```
snort -u snort -g snort -c /etc/snort/snort.conf &>
ids1_test1.txt
```

```
snort -u snort -g snort -c /etc/snort/snort.conf &>
ids2_test1.txt
```

NOTE: All of the output (stdout and stderr) will be piped to the text files and hold the command prompt. To view the text files to verify Snort properly initialized switch to another virtual console (Alt+F2) and search the test file for ‘Initialization Complete’

4. From the workstation that can access the RH8 web server, open IE or Firefox and browse to the secret BASE web site. Record the Total Number of Alerts on the home page or clear all alerts.

5. Switch to the Unclass BASE website and Record the Total Number of Alerts on the home page or clear all alerts.

NOTE: To clear alerts:

Click on the number next to the Total Number of Alerts:

Click on the drop down arrow next to {action} and select Delete alerts

Click on Entire Query button

6. Switch to the Attacker machine and start IDSWakeup

Verify Attacker machine is configured for unclassified network with respect to bridge (VMWARE) and IP address.

Set the IP address to 192.168.100.50

```
ifconfig eth0 192.168.100.50
```

idswakeup <source IP> <destination IP>

```
idswakeup 192.168.100.50 192.168.100.3
```

The goal is to attack the RH8 server not the IDS itself to verify the IDS is reading the traffic.

Test will take about 1 to 2 minutes to run.

7. Refresh the browser; verify that alerts are being recorded. When the IDSWakeup test completes, record the Total Number of Alerts observed on the BASE home page to the table below.

Table for recording results:

Total # Alerts	Unique Alerts	TCP%	UDP%	ICMP%	Portscan Traffic %

8. On IDS1 do a crtl+z then kill %<pid> to stop the Snort process. Open the file ids1_test1.txt and search for ‘Action Stats’. Verify that the entries under Action Stats match the number of Total Number Alerts recorded from step 7

```
Action Stats:
Alerts:
Logged:
```

One other stat to check is dropped packets. Search for ‘Packet Wire Totals’ and look for ‘Dropped’. Count should be zero.

9. Switch to the secret BASE web site. If the traffic is properly segregated then the Total Number of Alerts count should have remained the same.

10. Reconfigure the Attacker box to operate on the secret network. Reconfigure the bridge (VMWare) and IP address

```
ifconfig eth0 192.168.101.50
```

11. Restart the Snort process on IDS1 but do not pipe the output to a text file. Repeat the IDSWakeup test

```
snort -u snort -g snort  -c /etc/snort/snort.conf
```

```
idswakeup 192.168.101.50 192.168.101.3
```

12. Refresh the browser; verify that alerts are being recorded. When IDSWakeup is finished recorded the Total Number of Alerts observed in the table below.

Table for recording results:

Total # Alerts	Unique Alerts	TCP%	UDP%	ICMP%	Portscan Traffic %

13. On IDS2 do a crtl+z kill %<pid> to stop the Snort process. Open the file ids2_test1.txt and search for 'Action Stats'. Verify that the entries under Action Stats match the number of Total Number Alerts recorded from step 13

```
Action Stats:
Alerts:
Logged:
```

One other stat to check is dropped packets. Search for 'Packet Wire Totals' and look for 'Dropped'. Count should be zero.

14. Switch to the unclassified BASE web site. If the traffic is properly segregated then the Total Number of Alerts count should have remained the same.

C. XTS400 IDS ARCHITECTURE FUNCTIONAL TEST

1. Verify database and web site components in the XTS400 environment are running. Edit the Snort configuration file /etc/snort/snort.conf on IDS1 and IDS2 to send alerts to the PostgreSQL database on the XTS400 system.

Settings for IDS1:

```
output database: log, postgresql, user=snort
dbname=snort password=mysea host=192.168.100.130
port=5433
```

Settings for IDS2:

```
output datagase: log, postgresql, user=snort
```

```
dbname=snort password=mysea host=192.168.101.130
port=5434
```

2. Stop any running Snort processes on each IDS.

Start the Snort process on each IDS.

```
snort -u snort -g snort -c /etc/snort/snort.conf &>
ids1_test2.txt
```

```
snort -u snort -g snort -c /etc/snort/snort.conf &>
ids2_test2.txt
```

NOTE: All of the output (stdout and stderr) will be piped to the text files and hold the command prompt. To view the text files to verify Snort properly initialized switch to another virtual console (Alt+F2) and search the test file for ‘Initialization Complete’

3. From the workstation start the TCBE and then start a SIM_UNCLASSIFIED session as user mdemo1.

4. Open the unclassified BASE web site and record the Total Number of Alerts on the home page or clear all alerts. Close the web browser.

5. Change to a SIM_SECRET session and open the secret BASE web site.

6. The Attacker machine should still be configured for the secret network. Rerun the idswakeup test.

```
     idswakeup 192.168.101.50 192.168.101.130
```

7. Refresh the browser; verify that alerts are being recorded. When the IDSWakeup test completes, record the Total Number of Alerts observed from the secret BASE home page to the table below.

Table for recording results:

Total # Alerts	Unique Alerts	TCP%	UDP%	ICMP%	Portscan Traffic %

8. On IDS2 do a crtl+z kill %<pid> to stop the Snort process. Open the file ids2_test2.txt and search for 'Action Stats'. Verify that the entries under Action Stats match the number of Total Number Alerts recorded from step 8

```
Action Stats:
Alerts:
Logged:
```

One other stat to check is dropped packets. Search for 'Packet Wire Totals' and look for 'Dropped'. Count should be zero.

9. Close the browser.

10. Change to SIM_UNCLASSIFIED session then open the unclassified BASE web site. If the sensors are properly segregated then the Total Number of Alerts count should have remained the same.

11. Reconfigure the Attacker machine to operate on the unclassified network.

If using VMWare reconfigure the bridge

```
ifconfig eth0 192.168.100.50
```

12. Restart the Snort process on IDS2 but do not pipe the output to a file. Rerun the idswakeup test.

```
snort -u snort -g snort -c /etc/snort/snort.conf
idswakeup 192.168.100.50 192.168.100.130
```

13. Refresh the browser; verify that alerts are being recorded. When the IDSWakeup test completes, record the Total Number of Alerts observed from the unclassified BASE home page to the table below.

Table for recording results:

Total # Alerts	Unique Alerts	TCP%	UDP%	ICMP%	Portscan Traffic %

14. On IDS1 do a crtl+z kill%<pid> to stop the Snort process. Open the file ids1_test2.txt and search for ‘Action Stats’. Verify that the entries under Action Stats match the number of Total Number Alerts recorded from step 14

```
Action Stats:
Alerts:
Logged:
```

One other stat to check is dropped packets. Search for ‘Packet Wire Totals’ and look for ‘Dropped’. Count should be zero.

This completes the testing of both the Red Hat 8 and XTS400 environments. Stop all Snort processes and log out of all sessions.

LIST OF REFERENCES

[1] C. Irvine, T. Levin, T. Nguyen, D. Shifflet, J. Khosalim, P. Clark, A. Wong, F. Afinidad, D. Bibighaus, and J. Sears, "Overview of a High Assurance Architecture for Distributed Multilevel Security" in Proceedings 5th IEEE Systems, Man and Cybernetics Information Assurance Workshop, United States Military Academy, West Point, NY. 10 – 11 June 2004, pp. 38 – 45.

[2] J. Horn, IPSEC-BASED DYNAMIC SECURITY SERVICES FOR THE MYSEA ENVIRONMENT, June 2005, Naval Postgraduate School, Monterey, CA.

[3] M. Focke, BAE SYSTEMS XTS-400 Trusted Computer System Technical Overview, BAE Systems Information Technology, Herndon, VA.

[4] T. Kohlenberg, et. al., Snort IDS and IPS Toolkit, 2007, Syngress Publishing, Inc. Burlington, MA.

[5] Basic Analysis Security Engine (2008, February). Available: http://base.secureideas.net/. Accessed: (February/2008).

[6] J. Anderson, Computer Security Threat Monitoring and Surveillance, Contract 79F296400, Fort Washington, Pa, February 26, 1980.

[7] D. Denning, An Intrusion-Detection Model, IEEE Transactions of Software Engineering, Vol. SE-13, NO. 2, February 1987..

[8] MySQL Proxy Service (2008, February). Available: http://dev.mysql.com/tech-resources/articles/proxy-gettingstarted.html. Accessed: (February 2008).

[9] K. Scarfone, P. Mell, Guide to Intrusion Detection and Prevention Systems, Recommendations of the National Institute of Standards and Technology, NIST, Special Publication 800-94, Gaithersburg MD, February/2007.

[10] R. Ierusalimschy, et. al., Lua 5.1 Reference Manual, Lua.org, August 2006.

[11] Open Source Initiative, Available: http://www.opensource.org/, Accessed: (March/2008).

[12] Linuxworld, Available: http://www.linuxworld.com/news/2007/0301207-top-5-security.html, Accessed: (March/2008).

[13] Infosecwriters, Available: http://infosecwriters.com, Accessed: (March/2008).

[14] Snort, Available: http://www.snort.org, Accessed: (March/2008).

[15] Bro IDS, Available: http://www.bro-ids.org, Accessed: (March/2008).

[16] V. Paxson, et. al., Bro Quick Start Guide, version 0.9, 11-15-2004, The Regents of the University of California, 2004.

[17] C. Gosselin, Open Source Intrusion Detection and Prevention: Tools for Today's Corporate Market?, Eastern Carolina University.

[18] Open Source Database Review, Available: http://www.geocities.com/mailsoftware42/db/, Accessed: (March/2008).

[19] Aanval, Available http://www.aanval.com, Accessed: (March/2008).

[20] Basic Analysis Security Engine, Available: http://base.secureideas.net/, Accessed: (March/2008).

[21] SGUIL, Available: http://sguil.sourceforge.net/, Accessed: (March/2008).

[22] Executive Order 12958, Part 3 Declassification and Downgrading, Office of the Press Secretary, March 25, 2003.

[23] MySQL Internals Client Server Protocol, http://forge.mysql.com/wiki/MySQL_Internals_ClientServer_Protocol, Accessed: (April/2008).

[24] SRI International project EMERALD, http://www.sdl.sri.com/projects/emerald/concepts.html, Accessed: (May/2008).

[25] SRI International project EMERALD, http://www.sdl.sri.com/projects/emerald/project.html, Accessed: (May/2008).

[26] MySQL Proxy, http://dev.mysql.com/tech-resources/articles/proxy-gettingstarted.html, Accessed (May/2008).

[27] ADOdb, http://adodb.sourceforge.net/, Accessed (May/2008).

[28] Apache, http://httpd.apache.org/docs/1.3/dso.html, Accessed (May/2008).

[29] Insecure.org, http://seclists.org/bugtraq/2005/Apr/0305.html, Accessed (May/2008).

[30] Cert, http://www.cert.org/advisories/CA-1996-11.html, Accessed (May/2008).

[31] IDSWakeup, http://www.hsc.fr/ressources/outils/idswakeup/index.html.en, Accessed (May/2008).

[32] pg_shawdow, http://ou800doc.caldera.com/en/PostgresqlDoc/catalog-pg-shadow.html, Accessed (May/2008).

[33] PostgreSQL Database Cluster, http://www.postgresql.org/docs/7.4/static/creating-cluster.html, Accessed (May/2008).

[34] How to test Snort, R. Bejtlich, http://searchsecuritychannel.techtarget.com/tip/0,289483,sid97_gci1266313,00.html#, August 2007. Accessed (May/2008).

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Susan Alexander
   OASD/NII DOD/CIO
   Washington, D.C.

4. Hugo A. Badillo
   NSA
   Fort Meade, MD

5. George Bieber
   OSD
   Wasington, D.C.

6. John Cambell
   National Security Agency
   Fort Meade, MD

7. Deborah Cooper
   DC Associates, LLC
   Roslyn, VA

8. Dr. Grace Crowder
   NSA
   Fort Meade, MD

9. Louise Davidson
   National Geospatial Agency
   Bethesda, MD

10. Steve Davis
    NRO
    Chantilly, VA

11. Vincent J. DiMaria
    National Security Agency
    Fort Meade, MD

12. Dr. Tim Fossum
National Science Foundation
Arlington, VA

13. Jennifer Guild
SPAWAR
Charleston, SC

14. CDR Scott Heller
SPAWAR
Charleston, SC

15. Steve LaFountain
NSA
Fort Meade, MD

16. Dr. Greg Larson
IDA
Alexandria, VA

17. Dr. Karl Levitt
NSF
Arlington, VA

18. Dr. John Monastra
Aerospace Corporation
Chantilly, VA

19. John Mildner
SPAWAR
Charleston, SC

20. Jim Roberts
Central Intelligence Agency
Reston, VA

21. Ed Schneider
IDA
Alexandria, VA

22. Mark Schneider
NSA
Fort Meade, MD

23. Keith Schwalm
Good Harbor Consulting, LLC
Washington, DC

24. Ken Shotting
NSA
Fort Meade, MD

25. CDR Wayne Slocum
SPAWAR
San Diego, CA

26. Dr. Ralph Wachter
ONR
Arlington, VA

27. Matt Warnock
Booze-Allen-Hamilton
Arlington, VA

28. Dr. Cynthia E. Irvine
Naval Postgraduate School
Monterey, CA

29. Thuy D. Nguyen
Naval Postgraduate School
Monterey, CA

30. Thomas Tenhunen
SFS Student: Civilian, Naval Postgraduate School
Monterey, CA